Outsourcing Multi-Party Computation

Seny Kamara [*] Payman Mohassel [†] Mariana Raykova [‡]

Abstract

We initiate the study of secure multi-party computation (MPC) in a *server-aided* setting, where the parties have access to a *single* server that (1) does not have any input to the computation; (2) does not receive any output from the computation; but (3) has a vast (but bounded) amount of computational resources. In this setting, we are concerned with designing protocols that minimize the computation of the parties at the expense of the server.

We develop new definitions of security for this server-aided setting that generalize the standard simulation-based definitions for MPC and allow us to formally capture the existence of dishonest but *non-colluding* participants. This requires us to introduce a formal characterization of non-colluding adversaries that may be of independent interest.

We then design general and special-purpose server-aided MPC protocols that are more efficient (in terms of computation and communication) for the parties than the alternative of running a standard MPC protocol (i.e., without the server). Our main general-purpose protocol provides security when there is at least one honest party with input. We also construct a new and efficient server-aided protocol for private set intersection and give a general transformation from any secure delegated computation scheme to a server-aided two-party protocol.

[*]Microsoft Research. senyk@microsoft.com.

[†]University of Calgary. pmohassel@cspc.ucalgary.ca. Work done while visiting Microsoft Research.

[‡]Columbia University. mariana@cs.columbia.edu. Work done as an intern at Microsoft Research.

Contents

1 Introduction **3**
- 1.1 Our Contributions . . . 4
- 1.2 Overview of Protocols . . . 5

2 Related Work **7**

3 Preliminaries and Standard Definitions **8**

4 Non-Collusion in Multi-Party Computation **9**
- 4.1 Formalizing Non-Collusion With Respect to Semi-Honest Adversaries . . . 10
- 4.2 Formalizing Non-Collusion With Respect to Deviating Adversaries . . . 12

5 Efficiency in the Server-Aided Setting **15**
- 5.1 Evaluating the Efficiency Gain . . . 15
- 5.2 Comparison with Secure Delegated Computation . . . 16
- 5.3 Why Non-Collusion Helps . . . 16

6 An Efficient Protocol for Non-Colluding Semi-Honest Parties **17**

7 Protecting Against Deviating Circuit Garblers **20**
- 7.1 What goes wrong in the server-aided setting? . . . 21
- 7.2 Extending to Multiple Parties . . . 27

8 Server-Aided Computation From Delegated Computation **28**

9 Private Set Intersection in the Server-Aided Model **31**

References **36**

A Garbled Circuits **39**

B Secure Delegated Computation **40**

1 Introduction

It is often the case that mutually distrustful parties need to perform a joint computation but cannot afford to reveal their inputs to each other. This can occur, for example, during auctions, data mining, voting, negotiations and business analytics. Secure multi-party computation (MPC) allows a set of n parties, each with a private input, to securely and jointly evaluate an n-party functionality f over their inputs. Roughly speaking, an MPC protocol guarantees (at least) that (1) the parties will not learn any information from the interaction other than their output and what is inherently leaked from it; and (2) that the functionality was computed correctly.

Early feasibility results in MPC showed that any functionality can be securely computed [Yao82, Yao86, GMW87, CCD88]. Since then, there has been a large number of works improving the security definitions, strengthening the adversarial model, increasing the number of malicious parties tolerated and studying the round and communication complexity (we refer the reader to [Gol04] and the references therein). Most of these works, however, implicitly assume that the computation will be executed in a *homogeneous* computing environment where all the participants play similar roles, have the same amount of resources and, if corrupted, all collude with each other. In practice, however, computation does not always take place in such a setting. In fact, it is often the case that distributed computations are carried out by a diverse set of devices which could include high-performance servers, large-scale clusters, personal computers and even weak mobile devices. Furthermore, there are many instances in practice where collusion between participants is unlikely to occur. This can happen either because it is not feasible, is too costly, or because it is prevented by other means, e.g., by physical means, by the Law or due to conflicting interests. Non-collusion can also occur if the parties in the system are compromised by independent adversaries that simply do not have the capacity or the opportunity to compromise the same system (e.g., on one hand malware spreading through a network and on the other a dedicated attacker).

An important example of such a *heterogeneous* environment – and the main motivation for our work – is cloud computing, where a computationally powerful service provider offers (possibly weaker) clients access to its resources "as a service". In particular, we consider the problem of secure computation in a setting where, in addition to the parties evaluating the functionality, there is an *untrusted* server that (1) does not have any input to the computation; (2) does not receive any output from the computation; but (3) has a vast (but bounded) amount of computational resources. We refer to this setting as *server-aided* MPC and our aim is to design protocols that minimize the computation of the parties with inputs at the expense of the server.

Multi-party computation in the server-aided setting is interesting for several reasons. From a practical perspective, it can enable more efficient and scalable implementations of MPC. Until recently, MPC protocols were too inefficient to be used in practice but recent advances suggest that MPC could become practical [MNPS04, BDJ+06, LP07, LPS08, BDNP08, BCD+09, PSSW09]. Many of the improvements in efficiency, however, rely on new and more efficient instantiations of the cryptographic primitives and protocols underlying the general MPC results. While these improvements in efficiency are substantial, even the most efficient solutions cannot scale to handle the kind of massive datasets available today. Indeed, at a certain point, most algorithms (even non-secure ones) become impractical so the prospect of executing *any* MPC protocol is unreasonable. By allowing parties to securely outsource their computation to a cloud provider, server-aided MPC offers a different, but complimentary approach to making MPC practical and scalable.

In addition to its practical significance, server-aided MPC is also of theoretical interest. First, it is not a-priori clear that secure computation can be securely and *efficiently* outsourced to a *single* server. While fully-homomorphic encryption [Gen09] might seem like a natural approach, we note that: (1) it does not guarantee correctness of the computation; and (2) known constructions are currently not efficient enough for practice (especially not for large-scale data). Secure delegated computation [GGP10, CKV10, AIK10],

which allows a client to outsource computation to a single worker without revealing the input can also guarantee the correctness of the computation but known constructions only work for a *single* client and are not practical.

Finally, while MPC has been considered in a client/server model before [FKN94, IK97, NPS99, DI05, BCD+09], most previous work either requires (at least) two servers or does not provide any efficiency gain for the clients. One notable exception is the work of Feige, Killian and Naor [FKN94] which also considers a setting with a single server and which presents a protocol (from now on referred to as the FKN protocol) that allows the clients' work to be reduced at the expense of the server. Though this was not the original motivation of [FKN94], the setting considered in that work differs only slightly from ours and we show that, with minor modifications, the FKN protocol is a two-party server-aided protocol in our model.

1.1 Our Contributions

We initiate the study of MPC in the server-aided setting, where the parties have access to a *single* server with a large amount of computational resources that they can use to outsource their computation. In this setting, we are concerned with designing protocols that minimize the computation of the parties at the expense of the server. Server-aided MPC is well motivated in practice due to the increasing popularity of cloud computing and can enable practical and scalable MPC. We make several contributions:

1. We formalize and define security for server-aided MPC. Our security definition is in the ideal/real-world paradigm and guarantees that, in addition to the standard security properties of MPC, the server learns no information about the client's inputs or *outputs* and cannot affect the correctness of the computation.

2. We consider a new adversarial model for MPC in which corrupted parties do not necessarily collude. Non-collusion in heterogeneous computing environments often occurs in practice and therefore is important to consider. To address this question, we generalize the standard security definition for MPC to allow for a finer-grained specification of collusion between the parties. This requires us to introduce formal characterizations of non-colluding adversaries which may be of independent interest. Also, as we will see, by considering non-colluding adversaries we are be able to obtain highly efficient protocols.

3. We explore the connection between server-aided MPC and secure delegated computation. Roughly speaking, a server-aided MPC protocol can be viewed as a (interactive) delegated computation scheme for multiple parties. We show how to transform any secure delegated computation scheme into a server-aided MPC protocol.

In addition to the theoretical contributions discussed above, we also describe two efficient general-purpose server-aided MPC protocols based on Yao's garbled circuits [Yao82], and an efficient special-purpose protocol for *private set intersection* (which has been the subject of much recent work):

4. The first protocol we consider is (a slight variant of) the FKN protocol [FKN94]. We show that it is secure against a malicious server and semi-honest parties that do not collude with the server. It allows all but one of the parties to outsource their computation to the server; making their computation only linear in the size of their inputs (which is optimal). In addition, the protocol does not require any public-key operations (except for a one time coin-tossing protocol).

5. Our second protocol extends the FKN protocol to be secure even when all but one of the parties is malicious. Our construction uses cut-and-choose techniques from [MF06, LP07], but requires us to address several subtleties that do not exist in the standard two-party setting. One main problem we

need to solve is how to allow an *untrusted* server to send the majority output of multiple circuits to the parties without learning the actual output or modifying the results. We achieve this by constructing a new *oblivious* cut-and-choose protocol that allows the verifier in the cut-and-choose mechanism to outsource its computation to an *untrusted* server.

6. Our third protocol is a new and efficient server-aided protocol for private set intersection. Our construction provides security against a malicious server as well as malicious participants. The bulk of computation by each participant is a number of PRF evaluations that is linear in the size of its input set. In comparison, the most efficient two-party set intersection protocol [HN10] with equivalent security guarantees, requires $O(m + n(\log\log m + p))$ exponentiations where m and n are the sizes of the input sets and p is the bit length of each input element, which is logarithmic in the input domain range. Protocols that provide security in weaker adversarial models (e.g., the random oracle model, the CRS model, and limited input domain) and achieve linear computational complexity in the total size of the input sets still require a linear number of public key operations.

   Our solution generalizes to the case of multi-party set intersection, preserving the same computational complexity for each participant. The best existing solution for this case [DSMRY11] has computation cost of $O(Nd^2 \log d)$ in the case of N parties with input sets of size d.

The above-mentioned protocols are significantly more efficient than the existing MPC protocols for the same problems. In order to provide a better sense of the *efficiency gain* obtained by these server-aided constructions, we initiate in Section 5 an informal discussion on efficiency in the server-aided model. In particular we outline different ways of quantifying the efficiency of a server-aided MPC protocol, each of which is suitable for a specific setting or application. We also provide some intuition for why any noticeable improvement in the efficiency of our general-purpose constructions is likely to yield considerably more practical secure delegated computation schemes. We note that a more formal study of efficiency in the server-aided model is an interesting research direction.

1.2 Overview of Protocols

In this overview and throughout the rest of the paper, we mostly focus on server-aided two-party computation (two parties and a server). However, as we discuss in future sections, our constructions easily extend to the multi-party case as well.

The first two protocols are based on Yao's garbled circuit construction. In both protocols, the only *interaction* between the two parties, denoted by P_1 and P_2, is to generate a set of shared random coins to be used in the rest of the protocol. This step is independent of the parties' inputs and the function being evaluated and can be performed offline (e.g., for multiple instantiations of the protocol at once). Moreover, the coin-tossing needs to be performed exactly once to share a secret key. In all future runs of the protocol, P_1 and P_2 can use their shared secret key and a pseudorandom function, to generate the necessary coins. After this step, the two parties interact directly with the *untrusted server* until they retrieve their final result.

The FKN protocol. In the (modified) FKN protocol (described in Section 6), after generating the shared random coins, each party sends a single message to the server and receives an encoding of his own output. The parties then individually use their local coins to recover their outputs. For P_1 and P_2, this protocol is significantly more efficient than using a standard secure two-party computation protocol. First, none of the parties (including the server) have to perform any public-key operations, except to perform the coin-tossing which as discussed above is only performed once. This is in contrast to standard MPC where public-key operations (which are considerably more expensive than their secret-key counterparts)

are a necessary. Second, one of the parties (P_2 in our case) only needs to do work that is linear in the size of his input and output, and independent of the size of the circuit being computed.

The server and P_1 will do work that is linear in the size of the circuit. If the server-aided protocol is run multiple times for the same or different functionalities, it is possible to reduce the online work of P_1 by performing the garbling of multiple circuits (for the same or different functions) in the offline phase. The online work for P_1 will then be similar to P_2 and only linear to the size of his input and output.

We prove the protocol secure against a number of corruption and non-collusion scenarios. Particularly, the protocol can handle cases where the server, or P_2 are malicious while the other players are either honest, or semi-honest but non-colluding.

Making FKN robust. The security of the modified FKN protocol breaks down when party P_1 is malicious. We address this in Section 7 by augmenting it with cut-and-choose techniques for Yao's two-party protocol (e.g., see [MF06, LP07]). First, note that the cut-and-choose procedure no longer takes place between P_1 and P_2 since this would require P_2 to do work linear in the circuit size (significantly reducing his efficiency gain). Instead, P_2 outsources his cut-and-choose verification to the server. However, a few subtleties arise that are specific to the server-aided setting and require modifications to the existing techniques. At a high level, the difficulty is that, unlike the standard setting, the server only learns the garbled outputs and therefore cannot determine the majority output on his own. One might try to resolve this by having the server send the garbled outputs to P_1 and P_2 and have them compute the majority but, unfortunately, this is also insecure.

We take the following approach. Instead of treating the garbled output values as the garbling of the real outputs (as prescribed by the translation table), we use them as evaluation points on random polynomials that encode (as their constant factor) the "ultimate" garbled values corresponding to the real outputs. This encoding can be interpreted as a Reed-Solomon encoding of the ultimate garbled values using the intermediate ones returned by the circuit evaluation. Intuitively, as long as the majority of the garbled evaluation points are correct, the error correction of Reed-Solomon codes guarantees correct and unambiguous decoding of the majority output by the server. Further care is needed to ensure that the server performs the decoding obliviously, and that for each output bit he is only able to decode one Reed-Solomon codeword without learning the corresponding output. For this purpose, it turns out that we need the polynomials used for the Reed-Solomon encoding to be permutations over the finite field. To achieve this goal, we sample our polynomials uniformly at random from the family of Dickson polynomials of an appropriate degree.

Delegation-based protocol. We show how to construct a server-aided two-party computation protocol based on any secure non-interactive *delegated computation* scheme. A delegated computation scheme allows a client to securely outsource the computation of a circuit C on a private input x to an untrusted worker. The notion of secure delegated computation is closely related to that of verifiable computation [GKR08, GGP10, CKV10], with the additional requirement that the client's input and output remains private. Our construction is interesting in the sense that, it formalizes the intuitive connection between the problems of server-aided computation and verifiable computation by interpreting server-aided protocols as a means for verifiably and privately outsourcing a secure two-party computation protocol to an untrusted worker. The resulting protocol inherits the efficiency of the underlying delegated computation scheme.

Set intersection protocol. We present a construction for outsourced computation for the problem of set intersection where two or more parties want to find the intersection of their input sets. The main idea of our protocol is that to have the server compute the set intersection of the input sets but since we want to preserve the privacy of the inputs, he is only given PRF evaluations on the elements under a key that

all parties have agreed on. Then each party will be able to map the returned intersection PRF values to real elements. In order to protect against a malicious server, we augment this approach by mapping each input value to several unlinkable PRF evaluations. Now the server will be asked to compute the set intersection on the new expanded sets and will be able to cheat without being detected only if he guesses correctly which PRF evaluations correspond to the same input value. This approach, however, introduces a new security issue in that it allows the parties to be malicious by creating inconsistent PRF values. We fix this by requiring each party to prove that he has computed correctly the multiple PRF evaluations for each of his input elements by opening them after the server has committed to the output result. In order not to lose the privacy guarantees for the inputs, however, we apply another level of PRF evaluations.

2 Related Work

We already mentioned early feasibility results in MPC so we focus on work more closely related to our own.

Server-aided computation with a single server. Similar settings to our own have been considered in the past. Most notably, Feige, Killian and Naor [FKN94] consider a setting that includes two parties with private inputs and a server without input. In this setting, the parties send a single message to the server who then learns the output. The main differences between [FKN94] and our work are that in our setting: (1) interaction is allowed; (2) the server is not allowed to learn the output of the evaluation; (3) the server is not assumed to be semi-honest; and (4) we wish to outsource the computation of the parties at the expense of the server. Though our motivation is quite different from [FKN94], the settings vary only slightly and we show that one of the protocols proposed in [FKN94] can be used in our setting with only minor modifications. We recall the protocol in Section 6 and prove its security with respect to our new adversarial models and definitions.

Another similar setting is that of Naor, Pinkas and Sumner [NPS99] who propose a protocol based on Yao's garbled circuit construction [Yao82] for secure auctions. Here, an auction issuer prepares an auction to be executed between an auctioneer and a set of bidders. The protocol guarantees that the auctioneer and the auction issuer do not learn any information about the bids as long as they do not collude. This is similar to our setting in that the auctioneer has no input to the computation and evaluates the auction "on behalf" of the bidders and of the auction issuer. The main differences between the setting of [NPS99] and our own is that in our setting: (1) the server is not allowed to learn the output; and (2) the parties with input are not assumed to be semi-honest.

Server-aided computation with multiple servers. Server-aided computation with *multiple* servers has also been considered [DI05, BCD+09]. Similar to our own motivation, previous work in this multi-server setting is explicitly concerned with saving work for the clients at the expense of the servers. Catrina and Kerschbaum [CK08] suggest a setting in which there are additional parties that do not have any inputs but assist the execution in different ways, e.g., with coordination and communication, correctness of computation, or distributed trust among multiple additional parties. It is also noted that solutions where a single additional party could assist the computation in an efficient way is an open problem since most MPC protocols rely on secret sharing techniques.

Collusion in MPC. The problem of collusion in MPC was first explicitly considered in the work Lepinski, Micali and Shelat [LMs05], where they defined and gave constructions of collusion-free protocols. Roughly speaking, an MPC protocol is collusion-free if it meets all the standard security properties and,

in addition, it cannot be used as a covert channel. While the protocol of [LMs05] relies on physical assumption (e.g., ballot boxes and secure envelopes), recent work by Alwen, Shelat and Visconti [ASV08] and Alwen, Katz, Lindell, Persiano, Shelat and Visconti [AKL+09] shows how to construct collusion-free protocols that rely only on a trusted mediator. Unlike this line of work, our goal is not to design protocols that prevent collusion, but to formally define non-colluding parties for MPC.

Delegated computation. A verifiable computation scheme allows a client to verify the correctness of an outsourced computation performed by an untrusted worker. In the random oracle model, Micali's CS proofs [Mic94] can be used. In the standard model, Gennaro, Gentry and Parno propose an offline/online protocol [GGP10] based on Yao's garbled circuits and fully homomorphic encryption (FHE) [Gen09]. Their scheme requires the client to produce a public-key during the offline phase but has the added property that the worker learns nothing about the input. Chung, Kalai and Vadhan [CKV10] show how to remove the need for the public-key (though still making use of FHE) and suggests an interactive approach to delegate the offline phase using universal arguments. Applebaum, Ishai and Kushilevitz [AIK10] show how to efficiently convert the secrecy property of MPC protocols into soundness for a VC scheme via the use of randomizing polynomials and message authentication codes.

3 Preliminaries and Standard Definitions

Notation. We write $x \leftarrow \chi$ to represent an element x being sampled from a distribution χ, and $x \xleftarrow{\$} X$ to represent an element x being sampled uniformly from a set X. If f is a function, we refer to its domain as $\mathsf{Dom}(f)$ and to its range as $\mathsf{Ran}(f)$. The output x of an algorithm $\mathcal{A}$ is denoted by $x \leftarrow \mathcal{A}$. If $\mathcal{A}$ is a probabilistic algorithm we sometimes write $y \leftarrow \mathcal{A}(x; r)$ to make the coins r of $\mathcal{A}$ explicit. $[n]$ denotes the set $\{1, \ldots, n\}$. We refer to the ith element of a sequence $\overline{v}$ as either v_i or $\overline{v}[i]$. Throughout k will refer to the security parameter. A function $\nu : \mathbb{N} \to \mathbb{N}$ is negligible in k if for every polynomial $p(\cdot)$ and sufficiently large k, $\nu(k) < 1/p(k)$. Let $\mathsf{poly}(k)$ and $\mathsf{negl}(k)$ denote unspecified polynomial and negligible functions in k, respectively. We write $f(k) = \mathsf{poly}(k)$ to mean that there exists a polynomial $p(\cdot)$ such that for all sufficiently large k, $f(k) \leq p(k)$, and $f(k) = \mathsf{negl}(k)$ to mean that there exists a negligible function $\nu(\cdot)$ such that for all sufficiently large k, $f(k) \leq \nu(k)$.

Private key encryption. A private-key encryption scheme is a set of three polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ that work as follows. Gen is a probabilistic algorithm that takes a security parameter k in unary and returns a secret key K. Enc is a probabilistic algorithm that takes a key K and an n-bit message m and returns a ciphertext c. Dec is a deterministic algorithm that takes a key K and a ciphertext c and returns m if K was the key under which c was produced. Informally, a private-key encryption scheme is considered secure against chosen-plaintext attacks (CPA) if the ciphertexts it outputs do not leak any useful information about the plaintext even to an adversary that can adaptively query an encryption oracle.

Functionalities. An n-party randomized functionality is a function $f : (\{0,1\}^*)^n \times \{0,1\}^* \to \{0,1\}^*$, where the first input is a sequence of n strings $\overline{x}$, the second input is a set of random coins and the output is a sequence of n strings $\overline{y}$. We will often omit the coins and simply write $\overline{y} \leftarrow f(\overline{x})$. If we do wish to make the coins explicit then we write $\overline{y} \leftarrow f(\overline{x}; r)$. We denote the ith party's output by $f_i(\overline{x})$. A functionality is deterministic if it only takes the input string $\overline{x}$ as input and it is symmetric if all parties receive the same output. It is known that any protocol for securely computing deterministic functionalities can be used to securely compute randomized functionalities (cf. [Gol04] Section 7.3) so in this work we focus on the former. A basic functionality we will make use of is the coin tossing functionality $\mathcal{F}_{\mathsf{ct}}(1^\ell, 1^\ell) = (r, r)$, where $|r| = \ell$ and r is uniformly distributed.

Garbled circuits. Yao's garbled circuit construction consists of five polynomial-time algorithms $\mathsf{Garb} = (\mathsf{GarbCircuit}, \mathsf{GarbIn}, \mathsf{Eval}, \mathsf{GarbOut}, \mathsf{Translate})$ that work as follows. We present $\mathsf{GarbCircuit}$, GarbIn and $\mathsf{GarbOut}$ as deterministic algorithms that take a set of coins r as input. $\mathsf{GarbCircuit}$ is a deterministic algorithm that takes as input a circuit C that evaluates a function f, and a set of coins $r \in \{0,1\}^k$ and returns a garbled circuit $\mathrm{G}(C)$. GarbIn is a deterministic algorithm that takes as input a player index $i \in \{1,2\}$, an input x , coins $r \in \{0,1\}^k$, and returns a garbled input $\mathrm{G}(x)$. Eval is a deterministic algorithm takes as input a garbled circuit $\mathrm{G}(C)$ and two garbled inputs $\mathrm{G}(x)$ and $\mathrm{G}(y)$ and returns a garbled output $\mathrm{G}(o)$. $\mathsf{GarbOut}$ is a deterministic algorithm that takes as input a random coins $r \in \{0,1\}^k$ and returns a translation table T. $\mathsf{Translate}$ is a deterministic algorithm that takes as input a garbled output $\mathrm{G}(o)$ and a translation table T and returns an output o.

We note that it is possible to arrange the above five functions in such a way that the computational complexity of GarbIn is linear in the input size, the computational complexity of $\mathsf{GarbOut}$ and $\mathsf{Translate}$ is linear in the output size, while the complexity of $\mathsf{GarbCircuit}$ and Eval are linear in the circuit size. We use this important property when discussing efficiency of our protocols.

Informally, Garb is considered secure if $(\mathrm{G}(C), \mathrm{G}(x), \mathrm{G}(y))$ reveals no information about x and y. An added property possessed by the construction is *verifiability* which, roughly speaking, means that, given $(\mathrm{G}(C), \mathrm{G}(x), \mathrm{G}(y))$, no adversary can output some $\mathrm{G}(o)$ such that $\mathsf{Translate}(\mathrm{G}(o), \mathrm{T}) \neq f(x,y)$. We discuss these properties more formally in Appendix A.

Delegated computation. A delegated computation scheme consists of four polynomial-time algorithms $\mathsf{Del} = (\mathsf{Gen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ that work as follows. Gen is a probabilistic algorithm that takes as input a security parameter k and a function f and outputs a public and secret key pair $(\mathrm{PK}, \mathrm{SK})$ such that the public key encodes the target function f. $\mathsf{ProbGen}$ is a probabilistic algorithm that takes as input a secret key SK and an input x in the domain of f and outputs a public encoding σ_x and a secret state τ_x. $\mathsf{Compute}$ is a deterministic algorithm that takes as input a public key PK and a public encoding σ_x and outputs a public encoding σ_y. Verify is a deterministic algorithm that takes as input a secret key SK, a secret state τ_x and a public encoding σ_y and outputs either an element y of f's range or the failure symbol $\perp$. Informally, a delegated computation scheme is private if the public encoding σ_x of x reveals no useful information about x. In addition, the scheme is verifiable if no adversary can find an encoding $\sigma y'$, for some $y' \neq f(x)$, such that $\mathsf{Verify}_{\mathrm{SK}}(\tau_x, \sigma_{y'}) \neq \perp$. We say that a delegated computation scheme is secure if it is both private and verifiable. We provide formal definitions in Appendix B.

4 Non-Collusion in Multi-Party Computation

The standard ideal/real world definition for MPC, proposed by Canetti [Can01] and building on [Bea92, GL91, MR92], compares the real-world execution of a protocol for computing an n-party functionality f to the ideal-world evaluation of f by a trusted party. In the real-world execution, the parties run the protocol in the presence of an adversary $\mathcal{A}$ that is allowed to corrupt a subset of the parties. In the ideal execution, the parties interact with a trusted party that evaluates f in the presence of a simulator $\mathcal{S}$ that corrupts the same subset of parties.

Typically, only a single adversary $\mathcal{A}$ is considered. This *monolithic* adversary captures the possibility of collusion between the dishonest parties. One distinguishes between passive corruptions, where the adversary only learns the state of the corrupted parties; and active corruptions where the adversary completely controls the party and, in particular, is not assumed to follow the protocol. Typically, adversaries that make passive corruptions are *semi-honest* whereas adversaries that make active corruptions are *malicious*. In this work, we will make a distinction between malicious adversaries who make active corruptions and can behave arbitrarily and *deviating* adversaries who make active corruptions but whose behavior may

not be arbitrarily malicious (i.e., their behavior may be limited to a certain class of attacks). Another distinction can be made as to how the adversary chooses which parties to corrupt. If it must decide this before the execution of the protocol then we say that the adversary is *static*. On the other hand, if the adversary can decide during the execution of the protocol then we say that the adversary is *adaptive*. We only consider static adversaries in this work.

Roughly speaking, a protocol Π is considered secure if it emulates, in the real-world, an evaluation of f in the ideal-world. This is formalized by requiring that the joint distribution composed of the honest parties' outputs and of $\mathcal{A}$'s view in the real-world execution be indistinguishable from the joint distribution composed of the honest parties' outputs and the simulator $\mathcal{S}$'s view in the ideal-world execution. As mentioned above, the standard security definition for MPC models adversarial behavior using a *monolithic* adversary. This has the advantage that it captures collusion and thus provides strong security guarantees. There are, however, many instances in practice where collusion does not occur. This can happen either because it is not feasible, too costly, or because it is prevented by other means (e.g., by physical means, by the Law or due to conflicting interests). This is particularly true in the setting of cloud computing where one can think of many scenarios where the server (i.e., the cloud operator) will have little incentive to collude with any of the other parties.

This naturally leads to the two following questions: (1) how do we formalize secure computation in the presence of non-colluding adversaries? and (2) what do we gain by weakening the security guarantee? In particular, can we design protocols that are more efficient or that remain secure even if *all* parties are corrupted? Note that while the standard definition becomes meaningless if all parties are corrupted (since the monolithic adversary then knows all the private information in the system), this is not so if the parties are corrupted by non-colluding adversaries.

4.1 Formalizing Non-Collusion With Respect to Semi-Honest Adversaries

Intuitively, we think of collusion between participants in a MPC protocol as an exchange of "useful information", where by useful information we mean anything that allows the colluding parties to learn something about the honest parties' inputs that is not prescribed by the protocol. For our purposes, a non-colluding adversary is therefore:

> *"an adversary that avoids revealing any useful information to other parties."*

As we will see, formalizing this intuition is not straight-forward. In the following discussion, we divide the messages sent by a party into two types: *protocol* and *non-protocol* messages. Assuming the protocol starts with a "start" message and ends with an "end" message, a protocol message is one that comes after the start message and before the end message, and a non-protocol message is one that comes either before the start message or after the end message. To formalize our intuition we need to make two crucial changes to the standard definition which we discuss below.

Independent adversaries. First, in addition to the monolithic adversary (which corrupts multiple parties) we include a set of non-monolithic adversaries that corrupt at most one party and have access only to the view of that party. We also assume that all the adversaries are *independent* in the sense that they do not share any state [1]. Intuitively, this essentially guarantees that these adversaries do not send any non-protocol messages to each other and therefore that they can only collude using protocol messages. Throughout the rest of this work all adversaries are independent.

When working with independent adversaries, we will consider three possible adversarial behaviors: semi-honest, malicious and *non-cooperative*. Semi-honest and malicious behavior refer to the standard

[1] This idea already appears in work on collusion-free protocols [LMs05, ASV08, AKL+09].

notions: a semi-honest adversary follows the protocol while a malicious adversary can deviate arbitrarily. Informally, a non-cooperative adversary is one that deviates from the protocol as long as he does not (willingly) send useful information to another party. We will discuss how to formalize this intuition in section 4.2, and for now we focus on semi-honest adversaries. Note that, intuitively, if an adversary is independent and semi-honest, then it is non-colluding in the sense outlined above because it will send only protocol messages (by independence) and because these messages will reveal at most what is prescribed by the protocol (due its semi-honest behavior).

Partial emulation. Our second modification is a weakening of the notion of emulation to only require that indistinguishability hold with respect to the honest parties' outputs and a *single* adversary's view. In other words, we require that for each independent adversary $\mathcal{A}_i$, the joint distribution composed of the honest parties' outputs and $\mathcal{A}_i$'s view in the real-world, be indistinguishable from the joint distribution composed of the honest parties' outputs and the simulator $\mathcal{S}_i'$'s output in the ideal world. Roughly speaking, this implies that the protocol remains private (i.e., the parties do not learn information about each other's inputs) as long as the parties do not share any information.

To see why partial emulation is needed in our setting, consider two independent adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ whose outputs are correlated based on some protocol message exchanged between them. Under the standard notion of emulation, the simulators $\mathcal{S}_1$ and $\mathcal{S}_2$ would also have to output correlated views. The problem, however, is that because $\mathcal{S}_1'$ and $\mathcal{S}_2'$ are independent they cannot exchange any messages and a-priori it is not clear how they could correlate their outputs [2].

We are now ready to introduce our security definition for non-colluding semi-honest adversaries. Like the standard definition of security with abort, we do not seek to guarantee fairness or guaranteed output delivery and this is captured by allowing the adversaries to abort during the ideal-world execution. In addition, however, we also allow the server to select which parties will and will not receive their outputs. This weakening of the standard definition was first proposed by Goldwasser and Lindell in [GL02] and has the advantage of removing the need for a broadcast channel.

Real-world execution. The real-world execution of protocol Π takes place between players $(P_1, \ldots, P_n)$, server P_{n+1} and adversaries $(\mathcal{A}_1, \ldots, \mathcal{A}_{m+1})$, where $m \leq n$. Let $\mathrm{H} \subseteq [n+1]$ denote the honest parties, $\mathrm{I} \subset [n+1]$ denote the set of corrupted and non-colluding parties and $\mathrm{C} \subset [n+1]$ denote the set of corrupted and colluding parties. Since we only consider static adversaries these sets are fixed once the protocol starts.

At the beginning of the execution, each party $(P_1, \ldots, P_n)$ receives its input x_i and an auxiliary input z_i while the server P_{n+1} receives only an auxiliary input z_{n+1}. Each adversary $(\mathcal{A}_1, \ldots, \mathcal{A}_m)$ receives an index $i \in \mathrm{I}$ that indicates the party it corrupts, while adversary $\mathcal{A}_{m+1}$ receives C indicating the set of parties it corrupts.

For all $i \in \mathrm{H}$, let $\textsc{out}_i$ denote the output of P_i and for $i \in \mathrm{I} \cup \mathrm{C}$, let $\textsc{out}_i$ denote the view of party P_i during the execution of Π. The ith partial output of a real-world execution of Π between players $(P_1, \ldots, P_{n+1})$ in the presence of adversaries $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_{m+1})$ is defined as

$$\textsc{real}^{(i)}_{\Pi,\mathcal{A},\mathrm{I},\mathrm{C},\overline{z}}(k, \overline{x}) \stackrel{def}{=} \left\{\textsc{out}_j : j \in \mathrm{H}\right\} \cup \textsc{out}_i.$$

Ideal-world execution. In the ideal-world execution, all the parties interact with a trusted party that evaluates f. As in the real-world execution, the ideal execution begins with each party $(P_1, \ldots, P_n)$ receiving its input x_i and an auxiliary input z_i, while the server P_{n+1} receives only an auxiliary input z_{n+1}. Each party $(P_1, \ldots, P_n)$ sends x_i' to the trusted party, where $x_i' = x_i$ if P_i is semi-honest and x' is an arbitrary

[2] This could potentially be addressed using a setup assumption, but here we restrict ourselves to the plain model.

value if P_i is malicious. If any $x'_i = \perp$, then the trusted party returns $\perp$ to all parties. If this is not the case, then the trusted party asks the server to specify which of the corrupted parties should receive their outputs and which should receive $\perp$. The trusted party then returns $f_i(x'_1, \ldots, x'_n)$ to the corrupted parties P_i that are to receive their outputs and $\perp$ to the remaining corrupted parties. The trusted party then asks the corrupted parties that received an output whether they wish to abort. If any of them does, then the trusted party returns $\perp$ to the honest parties. If not, it returns $f_i(x'_1, \ldots, x'_n)$ to honest party P_i.

For all $i \in \mathrm{H}$, let $\textsc{out}_i$ denote the output returned to P_i by the trusted party, and for $i \in \mathrm{I} \cup \mathrm{C}$ let $\textsc{out}_i$ be some value output by P_i. The ith partial output of an ideal-world execution between parties $(P_1, \ldots, P_{n+1})$ in the presence of independent simulators $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_{m+1})$ is defined as

$$\textsc{ideal}^{(i)}_{f,\mathcal{S},\mathrm{I},\mathrm{C},\overline{z}}(k, \overline{x}) \stackrel{def}{=} \left\{\textsc{out}_j : j \in \mathrm{H}\right\} \cup \textsc{out}_i.$$

Security. Informally, a protocol Π is considered secure against non-colluding semi-honest adversaries if it *partially* emulates, in the real-world, an evaluation of f in the ideal-world.

Definition 4.1 (Security against semi-honest adversaries)**.** *Let f be a deterministic n-party functionality and Π be an n-party protocol. Furthermore, let $\mathrm{I} \subset [n+1]$ and $\mathrm{C} \subset [n+1]$ be such that $\mathrm{I} \cap \mathrm{C} = \emptyset$ and $|\mathrm{I}| = m$. We say that Π (I, C)-securely computes f if there exists a set $\{\mathbf{Sim}_i\}_{i \in [m+1]}$ of* PPT *transformations such that for all semi-honest* PPT *adversaries $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_{m+1})$, for all $\overline{x} \in (\{0,1\}^*)^n$ and $\overline{z} \in (\{0,1\}^*)^{n+1}$, and for all $i \in [m+1]$,*

$$\left\{\textsc{real}^{(i)}_{\Pi,\mathcal{A},\mathrm{I},\mathrm{C},\overline{z}}(k, \overline{x})\right\}_{k \in \mathbb{N}} \stackrel{c}{\approx} \left\{\textsc{ideal}^{(i)}_{f,\mathcal{S},\mathrm{I},\mathrm{C},\overline{z}}(k, \overline{x})\right\}_{k \in \mathbb{N}},$$

where $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_{m+1})$ and $\mathcal{S}_i = \mathbf{Sim}_i(\mathcal{A}_i)$.

4.2 Formalizing Non-Collusion With Respect to Deviating Adversaries

While non-collusion in the semi-honest model can be formalized via independent adversaries and partial emulation, this is not sufficient when the adversaries are allowed to deviate from the protocol. The difficulty is that such adversaries can use protocol messages to collude since they can send arbitrary messages. Of course, collusion can be prevented through physical means (e.g., using ballot boxes as in [LMs05]) or trusted communication channels (e.g., the mediator model used in [ASV08, AKL+09]) but our goal here is not to obtain (or define) protocols that prevent adversaries from colluding but to formally characterize adversaries that do not wish to or cannot collude.

Towards formalizing our intuition, we introduce the notions of *non-cooperative* and *isolated* adversaries. Let $(\mathcal{A}_1, \ldots, \mathcal{A}_{m+1})$ be a set of independent adversaries. Informally, an adversary $\mathcal{A}_i$ is non-cooperative with respect to another adversary $\mathcal{A}_j$ (for $j \neq i$) if it does not share any useful information with $\mathcal{A}_j$. An adversary $\mathcal{A}_i$ is isolated if all adversaries $\{\mathcal{A}_j\}_{j \neq i}$ are non-cooperative with respect to $\mathcal{A}_i$. In other words, $\mathcal{A}_i$ is isolated if no one wants to share any useful information with him. We formalize this intuition in the following definitions.

Definition 4.2 (Non-cooperative adversary)**.** *Let f be a deterministic n-party functionality and Π be an n-party protocol. Furthermore, let H, I and C be pairwise disjoint subsets of $[n+1]$ and let $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_{m+1})$, where $m = |\mathrm{I}|$, be a set of independent* PPT *adversaries. For any $i, j \in [m+1]$ such that $i \neq j$, we say that adversary $\mathcal{A}_j$ is non-cooperative with respect to $\mathcal{A}_i$ if there exists a* PPT *simulator $\mathcal{V}_{i,j}$ such that for all $\overline{x} \in (\{0,1\}^*)^n$ and $\overline{z} \in (\{0,1\}^*)^{n+1}$, and all $y \in \mathsf{Ran}(f_i) \cup \{\perp\}$,*

$$\left\{\mathcal{V}_{i,j}(y, z_i)\right\}_{k \in \mathbb{N}} \stackrel{c}{\approx} \left\{\mathsf{view}_{i,j} \;\middle|\; \mathsf{output}_i = y : \{\textsc{out}_\ell\}_\ell \leftarrow \textsc{real}^{(i)}_{\Pi,\mathcal{A},\mathrm{I},\mathrm{C},\overline{z}}(k, \overline{x})\right\}_{k \in \mathbb{N}}$$

whenever $\Pr[\,\mathsf{output}_i = y\,] > 0$. *Here* $\mathsf{view}_{i,j}$ *denotes the messages between* $\mathcal{A}_i$ *and* $\mathcal{A}_j$ *in the real-world execution and* $\mathsf{output}_i = y$ *is the event that party* P_i *receives output value* y.

Note that with the notion of non-cooperation, we are restricting the behavior of the non-cooperating adversary. In particular, we are assuming it will deviate from the protocol but in such a way that it will not disclose any useful information to the isolated adversary $\mathcal{A}_i$. Therefore, one has to be careful in specifying the isolated party's behavior (i.e., whether it is semi-honest or malicious) so that the non-cooperation assumption is not so strong as to imply the security of the protocol. In particular, requiring that the simulator $\mathcal{V}_{i,j}$ work with respect to a *malicious* $\mathcal{A}_i$ seems too strong. Similarly, requiring that it work with respect to an honest $\mathcal{A}_i$ seems too weak as honest adversaries can always be simulated. A more useful and reasonable notion seems to follow from requiring that $\mathcal{V}_{i,j}$ work with respect to *semi-honest* adversaries. This can be interpreted as saying that the non-cooperative adversary does not *intentionally* disclose useful information to an isolated adversary. In particular, this means that the non-cooperative adversary will not take actions such as sending its private input to the isolated party. It does not, however, restrict the isolated adversary from trying to "trick" the non-cooperative adversary into revealing this information.

As described above, an isolated adversary is one with which no other adversary wants to cooperate. Roughly speaking, we formalize this intuition by requiring that there exist an emulator for the isolated adversary $\mathcal{A}_i$ that, given only $\mathcal{A}_i$'s output value $f_i(\overline{x})$, returns $\mathcal{A}_i$'s view from a real-world execution. Intuitively, the notion of isolation restricts the behavior of the other adversaries towards $\mathcal{A}_i$ by allowing them to behave arbitrarily as long as their collective actions do not result in $\mathcal{A}_i$ learning any useful information in the sense discussed above. This informal description neglects some important subtleties that we address below.

Definition 4.3 (Isolated adversary)**.** *Let* f *be a deterministic* n*-party functionality and* Π *be an* n*-party protocol. Furthermore, let* H, I *and* C *be pairwise disjoint subsets of* $[n+1]$ *and let* $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_{m+1})$*, where* $m = |\mathrm{I}|$*, be a set of independent* PPT *adversaries. For any* $i, j \in [m+1]$ *such that* $i \neq j$*, we say that a semi-honest adversary* $\mathcal{A}_i$ *is isolated if there there exists a* PPT *emulator* $\mathcal{E}_i$ *such that for all* $\overline{x} \in (\{0,1\}^*)^n$ *and* $\overline{z} \in (\{0,1\}^*)^{n+1}$*, and all* $y \in \mathsf{Ran}(f_i) \cup \{\bot\}$*,*

$$\left\{\mathcal{E}_i\big(y, z_i\big)\right\}_{k\in\mathbb{N}} \overset{c}{\approx} \left\{\mathrm{OUT}_i \;\middle|\; \mathsf{output}_i = y : \{\mathrm{OUT}_\ell\}_\ell \leftarrow \mathrm{REAL}^{(i)}_{\Pi,\mathcal{A},\mathrm{I},\mathrm{C},\overline{z}}(k,\overline{x})\right\}_{k\in\mathbb{N}}$$

whenever $\Pr[\,\mathsf{output}_i = y\,] > 0$. *Here* $\mathsf{output}_i = y$ *is the event that party* P_i *receives output value* y.

Towards understanding Definition 4.3, it is perhaps instructive to consider how we will make use of it. Recall that to prove the security of a protocol Π against independent adversaries $(\mathcal{A}_1, \ldots, \mathcal{A}_{m+1})$, we need, for all $i \in [m+1]$, to describe a simulator $\mathcal{S}_i$ whose output in an ideal execution is indistinguishable from $\mathcal{A}_i$'s output in a real execution. To achieve this, we consider a simulator $\mathcal{S}_i$ that works by simulating $\mathcal{A}_i$ so that it can recover $\mathcal{A}_i$'s output and return it as its own. Now suppose that, in the real world, Π requires $\mathcal{A}_i$ to interact with some other adversary $\mathcal{A}_j$. It follows that $\mathcal{S}_i$ will somehow have to simulate this interaction, i.e., $\mathcal{S}_i$ will have to simulate the (protocol) messages from $\mathcal{A}_j$. These messages, however, could be "colluding messages" in the sense that they could carry information that helps $\mathcal{A}_i$ in learning more that what is prescribed by the protocol and it may be impossible for $\mathcal{S}_i$ to simulate them as they could include information known only to $\mathcal{A}_i$ and $\mathcal{A}_j$ (e.g., this information could be hardwired in $\mathcal{A}_i$ and $\mathcal{A}_j$). This is the main reason that simply requiring that the real and ideal adversaries be independent (together with partial emulation) is not enough to capture non-collusion with respect to deviating adversaries.

Our approach here will be to "strengthen" the simulator $\mathcal{S}_i$ by *assuming* that the adversaries $\{\mathcal{A}_j\}_{j\neq i}$ are non-cooperative with respect to $\mathcal{A}_i$. In Lemma 4.4 below, we will show that this implies that $\mathcal{A}_i$ is isolated and, therefore, there exists an emulator $\mathcal{E}_i$ that will return a view that is indistinguishable from $\mathcal{A}_i$'s view when interacting with $\{\mathcal{A}_j\}_{j\neq i}$ in a real-world execution.

Lemma 4.4. *Let f be a deterministic n-party functionality and Π be an n-party protocol. Furthermore, let $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_{m+1})$ be a set of independent* PPT *adversaries. For any $i, j \in [m+1]$, if $\{\mathcal{A}_j\}_{j \neq i}$ are non-cooperative with respect to $\mathcal{A}_i$, then $\mathcal{A}_i$ is isolated.*

Proof sketch: Consider the emulator $\mathcal{E}_i$ that, given y and z_i, computes $v_{i,j} \leftarrow \mathcal{V}_{i,j}(y, z_i)$ for all $j \neq i$ and returns the view OUT_i composed of $\{v_{i,j}\}_{i \neq j}$. Let $\{\mathcal{A}_{j_1}, \ldots, \mathcal{A}_{j_m}\}$ be the adversaries that are non-cooperative with respect to $\mathcal{A}_i$ and let $\{\mathcal{V}_{i,j_1}, \ldots, \mathcal{V}_{i,j_m}\}$ be their corresponding simulators (which are guaranteed to exist by the fact that they are non-cooperating with respect to $\mathcal{A}_i$). We show that $\mathcal{E}_i$'s output is indistinguishable from the view of $\mathcal{A}_i$ in a real execution using the following sequence of games. Game_0 consists of running $\{\text{OUT}_j\}_j \leftarrow \text{REAL}^{(i)}_{\Pi,\mathcal{A},\text{I},\text{C},\overline{z}}(k, \overline{x})$ and outputting OUT_i. For all $\ell \in [m]$, Game_ℓ is similar to $\mathsf{Game}_{\ell-1}$ except that the messages between $\mathcal{A}_i$ and $\mathcal{A}_{j_\ell}$ in OUT_i are replaced with messages generated by $\mathcal{V}_{i,j_\ell}(y, z_i)$. Note that the output of Game_m is distributed exactly as the output of $\mathcal{E}_i$.

The indistinguishability of the outputs of Game_0 and Game_1 follows directly from the non-cooperation of $\mathcal{A}_{j_1}$ with respect to $\mathcal{A}_i$ so we show that, for all $2 \leq \ell \leq m$, if there exists a PPT distinguisher $\mathcal{D}_\ell$ that distinguishes the output of Game_ℓ from that of $\mathsf{Game}_{\ell-1}$, then there exists a PPT distinguisher $\mathcal{B}_\ell$ that distinguishes the messages exchanged between $\mathcal{A}_i$ and $\mathcal{A}_{j_\ell}$ in a real-world execution from the messages generated by $\mathcal{V}_{i,j_\ell}$. Given a set of messages v_{i,j_ℓ} (generated either from a real-world execution or from $\mathcal{V}_{i,j_\ell}$), $\mathcal{B}_\ell$ works as follows. It first runs $\{\text{OUT}_j\}_j \leftarrow \text{REAL}_{\Pi,\mathcal{A},\text{I},\text{C},\overline{z}}(k, \overline{x})$ and, for $1 \leq t \leq \ell - 1$, it replaces the messages between $\mathcal{A}_i$ and $\mathcal{A}_{j_t}$ in OUT_i by $v_{i,j_t} \leftarrow \mathcal{V}_{i,j_t}(y, z_i)$. It then replaces the messages between $\mathcal{A}_i$ and $\mathcal{A}_{j_\ell}$ in OUT_i with v_{i,j_ℓ} and simulates $\mathcal{D}(\text{OUT}_i)$. It returns "real" if $\mathcal{D}$ outputs ℓ and "simulated" if $\mathcal{D}$ outputs $\ell+1$. Notice that if v_{i,j_ℓ} was generated from a real-world execution then $\mathcal{B}$ constructs OUT_i exactly as in $\mathsf{Game}_{\ell-1}$ whereas if v_{i,j_ℓ} is generated from $\mathcal{V}_{i,j_\ell}$ then $\mathcal{B}$ constructs OUT_i as in Game_ℓ. It follows then that $\mathcal{D}$'s advantage is equal to $\mathcal{B}$'s advantage in distinguishing between the outputs of $\mathsf{Game}_{\ell-1}$ and Game_ℓ which, by our initial assumption, is non-negligible.

■

Since, in our setting, we will consider multiple adversaries with different adversarial behaviors it will be convenient to specify the behavior of each adversary using the following notation. If $\mathcal{A}_1$ is semi-honest we will write $\mathcal{A}_1[sh]$ and if $\mathcal{A}_1$ is malicious we write $\mathcal{A}_1[m]$. If $\mathcal{A}_1$ is non-cooperative with respect to $\mathcal{A}_2$ then we write $\mathcal{A}_1[nc_2]$. Throughout, we will often need to describe classes of adversarial behaviors. We refer to such classes as *adversary structures* and describe them as follows. Consider, for example, a three party protocol between players P_1, P_2 and P_3. A protocol with security against the adversary structure

$$\mathsf{ADV} = \left\{ \left(\mathcal{A}_1[m], \mathcal{A}_2[m], \mathcal{A}_3[sh] \right), \left(\mathcal{A}_1[sh], \mathcal{A}_2[sh], \mathcal{A}_3[nc_1, nc_2] \right) \right\},$$

is secure in the following two cases: (1) $\mathcal{A}_1$ and $\mathcal{A}_2$ are malicious while $\mathcal{A}_3$ is semi-honest; and (2) $\mathcal{A}_1$ and $\mathcal{A}_2$ are semi-honest while $\mathcal{A}_3$ is non-cooperating with respect to both $\mathcal{A}_1$ and $\mathcal{A}_2$.

We now present our security definition for non-colluding deviating adversaries. It is based on the real- and idea-world executions defined in section 4.

Definition 4.5 (Security against deviating adversaries)**.** *Let f be a deterministic n-party functionality and Π be an n-party protocol. Furthermore, let $\text{I} \subset [n+1]$ and $\text{C} \subset [n+1]$ be such that $\text{I} \cap \text{C} = \emptyset$ and $|I| = m$ and let ADV be an adversary structure. We say that Π $(\text{I}, \text{C}, \mathsf{ADV})$-securely computes f if there exists a set $\{\mathbf{Sim}_i\}_{i \in [m+1]}$ of* PPT *transformations such that for all* PPT *adversaries $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_{m+1})$ that satisfy ADV, for all $\overline{x} \in (\{0,1\}^*)^n$ and $\overline{z} \in (\{0,1\}^*)^{n+1}$, and for all $i \in [m+1]$,*

$$\left\{ \text{REAL}^{(i)}_{\Pi,\mathcal{A},\text{I},\text{C},\overline{z}}(k, \overline{x}) \right\}_{k \in \mathbb{N}} \overset{c}{\approx} \left\{ \text{IDEAL}^{(i)}_{f,\mathcal{S},\text{I},\text{C},\overline{z}}(k, \overline{x}) \right\}_{k \in \mathbb{N}}$$

where $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_{m+1})$ and $\mathcal{S}_i = \mathbf{Sim}_i(\mathcal{A}_i)$.

Notice that the standard security definitions of secure MPC in the presence of a semi-honest and malicious adversary can be recovered from Definition 4.5 by setting $\mathrm{I} = \emptyset$ and letting the adversary in ADV be semi-honest or malicious.

5 Efficiency in the Server-Aided Setting

Having introduced the server-aided model for secure computation, it is natural to ask:

> *Is it possible to gain efficiency in this setting? and if so how can one quantify such a gain in efficiency?*

In this section we discuss these issues, informally. Formalizing these discussions accompanied with feasibility and impossibility results is an interesting direction for future research.

5.1 Evaluating the Efficiency Gain

Our main motivation for considering MPC in the server-aided setting is to allow (possibly) heterogeneous parties to outsource their computation to a server (that they do not necessarily trust). Hence, a natural way of measuring the efficiency of a server-aided protocol Π_f^{SAC} between n parties $(P_1, \ldots, P_n)$ and a server P_{n+1} that wish to compute a function f, is to compare the work performed by these parties with the work they would have to do in the most efficient "standard" MPC protocol Π_f^{MPC} (where the server is not present). Even if given a secure MPC protocol Π_f^{MPC} as a point of reference, there are multiple ways of quantifying the gain in the server-aided model each of which might be suitable for a particular computing environment.

Max/min efficiency. In cases where one party has significantly lower computational resources (e.g., a mobile phone executing a protocol with high-performance servers) we would like the weakest device to outsource as much of its computation as possible. A natural measure then is to consider only the maximum (over the parties) efficiency achieved by Π_f^{SAC}. Furthermore, when comparing to a standard MPC protocol, one should compare the total work of the party P_i that does the least amount of work in Π_f^{MPC} with the total work of the party P_j that does the least amount of work in Π_f^{SAC}. If the gap is significant enough, the weak device will see an improvement in efficiency if it plays the role of P_j in Π_f^{SAC} [3]. Similarly, one could consider the minimum efficiency, i.e., the total work of the party that does the most amount of work in Π_f^{SAC}.

We note that a sizable maximum efficiency might mean a less impressive reduction in the work of other players (or even an increase). For example, in the server-aided variant of Yao's garbled circuit protocol we design in Section 6, one player's computation is reduced significantly (the work is independent of the circuit size), while the second player has to do work proportional to the circuit size (though his work is still less than than what it would be in Yao's original protocol).

Average efficiency. Alternatively, one might be interested in a noticeable gain in the *total* work required by the players (excluding the server) regardless of the way this work is divided between the players. In this case one should try to optimize the *average* efficiency over by all the parties. To ensure some level of fairness, one can additionally enforce a limit on the variance in the efficiency of different parties. One example of a protocol with better average efficiency compared to standard two-party computation is briefly mentioned in Section 7. This protocol provides security against a malicious circuit garbler by utilizing the honest server for verification. In the resulting protocol, both players still have to compute a garbled

[3] We note that in the case of special-purpose protocols, the roles that a party can play may be restricted.

circuit once, but can avoid the cut-and-choose and/or zero-knowledge proof techniques which would add a significant overhead.

Combined efficiency. Finally, there may be cases where a combination of different efficiency measures may be appropriate. Consider, for example, a computation that occurs between a mobile device and a server running "in the cloud". As discussed above, the maximum efficiency of Π_f^{SAC} is an important consideration for the weakest device (i.e., the mobile device). But the minimum efficiency of Π_f^{SAC} may be important as well since computation "in the cloud" has an economic cost and the client may have a limited budget.

More generally, taking extra costs into account (e.g., the cost of a cloud service) for some players it may only make sense (economically) to take part in a server-aided protocol if the gain in efficiency is greater than a threshold while other players may be happy with more modest gains. In such cases a combination of the maximum, minimum and average efficiency might be appropriate.

5.2 Comparison with Secure Delegated Computation

An alternative way of measuring efficiency is to compare the work the parties have to perform in the server-aided protocol to their work if they were to take part in an *insecure* protocol for evaluating the same function. This measure of efficiency is closely related to those considered for *secure delegated computation* (see Appendix B). In fact, it is not hard to show that any server-aided protocol for computing a function f that achieves reasonable efficiency compared to an insecure protocol for the same task can easily be turned into an efficient and secure delegated computation.

For example, consider our server-aided variant of Yao's protocol from Section 6. As mentioned above, this protocol reduces P_2's work significantly, making it independent of the circuit size for f while P_1's work is still linear in the circuit size. Now consider a construction that would improve our protocol by making P_1's work sublinear in the circuit size. One could then transform P_1 and P_2 into a single client, and let the server be the worker in a securely delegated computation scheme. This would yield a scheme that is more efficient than existing *general-purpose* secure delegation schemes in the literature [GGP10, CKV10] since they all take advantage of heavy machinery (e.g., fully homomorphic encryption) and only provide *amortized* improvements over insecure protocols.

This connection with secure delegated computation schemes is bi-directional. In fact in Section 8, we formally show how to transform any secure delegated computation scheme into an efficient server-aided secure computation protocol.

5.3 Why Non-Collusion Helps

Our last note on efficiency is an intuitive explanation of why non-collusion helps. In particular, consider a security definition for server-aided computation where a player (e.g., P_1) could collude with the server S. In that case, we could combine P_1 and S into a single party P_{1S} and our security definitions would reduce to the standard definitions for secure two-party computation between P_{1S} and P_2. Then, any general-purpose protocol for server-aided computation where P_2's work is sublinear in the circuit size, would automatically yield a general-purpose secure two-party computation where one of the parties performs sublinear work in the size of circuit. To the best of our knowledge, the only known way of achieving this goal (even in the case of semi-honest adversaries) is via use of a fully-homomorphic encryption scheme.

On the other hand, our non-collusion assumption between P_1 and S allows us to design a general-purpose protocol where the work of P_2 is independent of the circuit size.

Inputs: P_1's input is x, P_2's input is y, and S has no inputs. All parties know the circuit C which computes the functions f.
Outputs: P_1 and P_2 learn the output $f(x, y)$.

1. P_1 and P_2 execute a coin tossing protocol to generate coins r. As a result, both players learn r.
2. P_1 computes $\mathrm{G}(C) \leftarrow \mathsf{GarbCircuit}(C; r)$, $\mathrm{G}(x) \leftarrow \mathsf{GarbIn}(C, 1, x; r)$ and $\mathrm{T} \leftarrow \mathsf{GarbOut}(r)$. It sends $\mathrm{G}(C)$ and $\mathrm{G}(x)$ to S.
3. P_2 computes $\mathrm{G}(y) \leftarrow \mathsf{GarbIn}(C, 2, y; r)$ and $\mathrm{T} \leftarrow \mathsf{GarbOut}(r)$. It then sends $\mathrm{G}(y)$ to S.
4. S computes $\mathrm{G}(o) \leftarrow \mathsf{Eval}(\mathrm{G}(C), \mathrm{G}(x), \mathrm{G}(y))$ and sends it to P_1 and P_2.
5. P_1 and P_2 separately compute $o \leftarrow \mathsf{Translate}(\mathrm{G}(o), \mathrm{T})$.

Figure 1: The (modified) FKN protocol.

6 An Efficient Protocol for Non-Colluding Semi-Honest Parties

In this Section, we describe a slight variation of the protocol by Feige, Killian and Naor from [FKN94] (from now referred to as the FKN protocol). The protocol makes use of Yao's garbled circuit construction as a black-box. We provide a high level description of Yao's construction in Appendix A.

At a high level, the FKN protocol works as follows. P_1 and P_2 are assumed to share a set of random coins. P_1 then uses these coins to generate a garbling of the circuit, the translation table and a garbling of its own input. P_1 sends the garbled circuit, its garbled input and the translation table to the server. P_2 uses the same coins, but only computes its own garbled input and the translation table. P_2 sends its garbled inputs to the server. The server evaluates the garbled circuit using the garbled inputs, translates the garbled output and returns the evaluation to both parties.

We slightly modify the FKN protocol to adapt it to our setting in which the server is not allowed to learn the output and where we do not assume the parties share a set of random coins. For this purpose, it suffices that (1) the parties execute a coin tossing protocol to generate the random coins; and (2) that P_1 not send the translation table to the server. The server can still evaluate the garbled output which the parties can translate on their own. Intuitively, the privacy and verifiability properties of the garbled circuit construction (see Appendix A) and the coin-tossing protocol guarantee that a malicious server cannot return the wrong result or learn anything about the inputs of P_1 and P_2 if he does not collude with either party.

We formally describe this variant of the FKN protocol in Figure 1 and in Theorem 6.2 below we show that it is secure against the following adversary structure:

$$\begin{aligned}\mathsf{ADV}_1 = \Big\{ &\Big(\mathcal{A}_S\big[sh\big], \mathcal{A}_1\big[sh\big], \mathcal{A}_2\big[sh\big]\Big),\\ &\Big(\mathcal{A}_S\big[m\big], \mathcal{A}_1\big[h\big], \mathcal{A}_2\big[h\big]\Big),\\ &\Big(\mathcal{A}_S\big[nc_1, nc_2\big], \mathcal{A}_1\big[sh\big], \mathcal{A}_2\big[sh\big]\Big),\\ &\Big(\mathcal{A}_S\big[h\big], \mathcal{A}_1\big[h\big], \mathcal{A}_2\big[m\big]\Big),\\ &\Big(\mathcal{A}_S\big[sh\big], \mathcal{A}_1\big[sh\big], \mathcal{A}_2\big[nc_S\big]\Big)\Big\}.\end{aligned}$$

Before proving the security of the FKN protocol in our model, we present a simple Lemma that we will use throughout this work and that will simplify our proofs significantly.

Lemma 6.1. *If a multi-party protocol* Π *between* n *players* $P_1, \ldots, P_n$*, securely computes* $f = (f_1, \ldots, f_n)$*,* (1) *in presence of semi-honest and independent parties and* (2) *in presence of a malicious* P_j *and honest* P_k *for* $k \in [n] - \{j\}$*, then the protocol is also secure in presence of* (3) *a* P_j *who is non-cooperative with respect to all other parties who are semi-honest.*

Proof. We need to prove that protocol Π is secure when the adversary A_j corrupting P_j is non-cooperative with respect to all other parties. Since A_j is non-cooperative, and all other parties are semi-honest (and hence non-cooperative), based on Lemma 4.4, we can assume that A_k is isolated for $k \in [n] - \{j\}$. Hence, we can use the definition of isolation in our simulation of such A_k's.

We first need to provide a simulator $\mathbf{Sim}_j$ for simulating a non-cooperative P_j in the ideal world. However, since Π is secure against a malicious P_j when all other parties are honest, we already know that a simulator $\mathbf{Sim}_{j'}$ exists that simulates P_j in that case. $\mathbf{Sim}_j$ imitates $\mathbf{Sim}_{j'}$ completely.

We also need to describe a simulator $\mathbf{Sim}_k$ for $k \in [n] - \{j\}$. The simulator $\mathbf{Sim}_k$ runs the adversary $\mathcal{A}_k$ controlling P_k in the real world, on input x_k. It sends x_k to the trusted party and receives back $f_k(x_1, \ldots, x_n)$. Since $\mathcal{A}_k$ is isolated, according to Definition 4.3, there exists an emulator $\mathcal{E}_k$ that takes $f_k(x_1, \ldots, x_n)$ as input and can be used to simulate a semi-honest $\mathcal{A}_k$'s output. $\mathbf{Sim}_k$ feeds $f_k(x_1, \ldots, x_n)$ to $\mathcal{E}_k$ and plays the role of semi-honest P_k in interaction with it. At the end of this interaction, $\mathbf{Sim}_k$ outputs what the semi-honest P_k would, and halts. According to the Definition 4.3, $\mathbf{Sim}_k$ will successfully simulate the output of $\mathcal{A}_k$ in this way.

■

□

Since we present all our proofs in the $\mathcal{F}_{\mathsf{ct}}$-hybrid model, we need to make sure that a coin-tossing protocol with security in all adversary structures we consider in fact exists. However, any two-party coin-tossing protocol (between P_1 and P_2) with security against malicious adversaries would be sufficient in our server-aided setting. Such a protocol can easily be proven secure when all three parties are semi-honest and independent. It is also secure, by definition, when either P_1 or P_2 are malicious. It is also secure when the server is malicious and the other two parties are honest since the server is not involved in the protocol, in any way. Finally, Lemma 6.1 guarantees that the same coin-tossing protocol is also secure in all adversary structures where one party is malicious and the rest are semi-honest and isolated.

We are now ready to state and prove the security of the FKN protocol with respect to ADV_1.

Theorem 6.2. *The (modified) FKN protocol described in Figure 1 securely computes any function* f *in the* $\mathcal{F}_{\mathsf{ct}}$*-hybrid model for the adversary structure* ADV_1*.*

Proof. We consider each case in ADV_1 separately.

Claim. The protocol $\big(\mathcal{A}_S[sh], \mathcal{A}_1[sh], \mathcal{A}_2[sh]\big)$-securely computes f in the $\mathcal{F}_{\mathsf{ct}}$-hybrid model.

We describe three independent transformations $\mathbf{Sim}_S$, $\mathbf{Sim}_1$ and $\mathbf{Sim}_2$:

- $\mathbf{Sim}_S$ simulates $\mathcal{A}_S$ as follows: it computes $(st, \mathrm{G}(C), \mathrm{T}) \leftarrow \mathsf{GarbCircuit}(C)$, $\mathrm{G}(x') \leftarrow \mathsf{GarbIn}(st, C, 1, x')$ and $\mathrm{G}(y') \leftarrow \mathsf{GarbIn}(st, C, 2, y')$ for random x' and y'; and sends $\mathrm{G}(C)$, $\mathrm{G}(x')$ and $\mathrm{G}(y')$ to $\mathcal{A}_S$. If $\mathcal{A}_S$ outputs $\perp$, then $\mathbf{Sim}_S$ tells the trusted party to abort. In either case, $\mathbf{Sim}_S$ outputs $\mathcal{A}_S$'s entire view.

  The privacy property of garbled circuits (see Definition A.2) guarantees that $\mathrm{G}(x)$ and $\mathrm{G}(y)$ are indistinguishable from x' and y' to $\mathcal{A}_S$ who does not know the coins r. In addition, in both the real

and ideal execution the semi-honest $\mathcal{A}_S$ does not abort since he is given valid garbled inputs (in the real world this is true since the other two parties are also semi-honest). Therefore, the views of $\mathcal{A}_S$ in the real and the ideal executions are indistinguishable.

- $\mathbf{Sim}_1$ receives x as input and sends it to the trusted party in order to receive $f(x, y)$. It then simulates $\mathcal{A}_1$ as follows. It answers $\mathcal{A}_1$'s $\mathcal{F}_{\mathsf{ct}}$ query by returning random coins r. $\mathbf{Sim}_1$ then computes $(st, \mathrm{G}(C), \mathrm{T}) \leftarrow \mathsf{GarbCircuit}(C; r)$ and uses the translation table to find a garbling $\mathrm{G}(o)$ of $f(x, y)$. Finally, it returns $\mathrm{G}(o)$ to $\mathcal{A}_1$ and outputs $\mathcal{A}_1$'s entire view.

  The view of $\mathcal{A}_1$ consists of the garbled circuits it creates and the garbled outputs it receives. In both the real and the ideal execution he receives the garbled output values corresponding to $f(x, y)$. In the real world this is guaranteed by the fact that S and P_2 are honest and the correctness property of garbled circuits (see Definition A.1). Therefore, the views of $\mathcal{A}_1$ in the real and the ideal executions are indistinguishable.

- $\mathbf{Sim}_2$ works analogously to $\mathbf{Sim}_1$.

□

Claim. The protocol $\big(\mathcal{A}_S[m], \mathcal{A}_1[h], \mathcal{A}_2[h]\big)$- securely computes f in the $\mathcal{F}_{\mathsf{ct}}$-hybrid model.

Consider the simulator $\mathbf{Sim}_S$ that simulates $\mathcal{A}_S$ as follows. It chooses coins r and computes $\mathrm{G}(C) \leftarrow \mathsf{GarbCircuit}(C; r)$. $\mathbf{Sim}_S$ chooses random inputs x' for P_1 and y' for P_2. Then he sends $\mathrm{G}(C)$ together with garbled input labels $\mathrm{G}(x') \leftarrow \mathsf{GarbIn}(C, 1, x'; r)$ and $\mathrm{G}(y') \leftarrow \mathsf{GarbIn}(C, 2, y'; r)$ to $\mathcal{A}_S$. $\mathbf{Sim}_S$ receives the garbled outputs that $\mathcal{A}_S$ returns for P_1 and P_2. If any of the outputs does not correspond to the correct value, the simulator instructs the trusted party to return $\perp$ to that party. The view of $\mathcal{A}_S$ consists of the garbled circuits and the garbled input values that he receives. The garbled values that correspond to zero and one are indistinguishable for the adversary since he does not know the seed for the PRG (correctness property in definition A.1). Therefore the garbled labels for the real inputs x and y in the real execution and the random values x' and y' in the ideal execution are indistinguishable for the $\mathcal{A}_S$. It follows the views of the adversary in the real and the ideal execution are also indistinguishable. The outputs of P_1 and P_2 are also indistinguishable in the real and the ideal execution. They receive the correct output, if $\mathcal{A}_S$ computes and returns the result honestly. Otherwise, in the ideal execution they receive $\perp$ from the trusted party, and in the real execution $\mathcal{A}_S$ cannot produce with all but negligible probability garbled output values for any other output but the correct evaluation of the garbled circuit by the verifiability property of garbled circuits (see Definition A.3).

□

Claim. The protocol $\big(\mathcal{A}_S[nc_1, nc_2], \mathcal{A}_1[sh], \mathcal{A}_2[sh]\big)$-securely computes f in the $\mathcal{F}_{\mathsf{ct}}$-hybrid model.

The proof of this claim is automatically implied given the last two claims and Lemma 6.1.

□

Claim. The protocol $\big(\mathcal{A}_S[h], \mathcal{A}_1[h], \mathcal{A}_2[m]\big)$-securely computes f in the $\mathcal{F}_{\mathsf{ct}}$-hybrid model.

$\mathbf{Sim}_2$ simulates the view of adversary $\mathcal{A}_2$. The simulator answers $\mathcal{A}_2$'s query to $\mathcal{F}_{\mathsf{ct}}$ with random coins r. He receives the garbled values $\mathrm{G}(y) \leftarrow \mathsf{GarbIn}(C, 2, y; r)$ corresponding to the input of $\mathcal{A}_2$. Using the coins r, $\mathbf{Sim}_2$ extracts $\mathcal{A}_2$'s input value y. If extraction fails he returns $\perp$ to $\mathcal{A}_2$. Otherwise, the simulator obtains the output $f(x, y)$ from the trusted party, computes the corresponding garbled output values and

sends them to $\mathcal{A}_2$. The view of $\mathcal{A}_2$, which consists of the output he receives, is indistinguishable in the real and the ideal execution. If the garbled input is contructed correctly, in both cases he gets $f(x, y)$ (in the real world this is true since the other two parties are honest). If he submits invalid garbled values for his input, he receives $\perp$ from the simulator in the ideal execution, and in the real world with high probability S will fail to evaluate the circuit and will return $\perp$. Similarly, the output of P_1 will be indistinguishable in the real and the ideal execution.

□

Claim. The protocol $\left(\mathcal{A}_S[sh], \mathcal{A}_1[sh], \mathcal{A}_2[nc_S]\right)$-securely computes f in the $\mathcal{F}_{\mathsf{ct}}$-hybrid model.

Once again the above claim is automatically implied given the proof of previous claims and Lemma 6.1.

□

■

Efficiency. First note that neither P_1, P_2 nor the server S have to perform any public key operations (i.e. no oblivious transfers are needed) with the exception of the initial coin-tossing between P_1 and P_2 which either requires the existence of a secure channel between them or public-key operations. Moreover, the coin-tossing needs to be performed exactly once to share a secret key. In all future runs of the protocol, P_1 and P_2 can use their shared secret key and a pseudorandom function, to generate the necessary seeds. This is a considerable improvement since public key operations are significantly more expensive compared to secret-key ones. Second, P_2 only needs to do work that is linear in the size of his input and the output, and independent of the circuit size since he only computes the $\mathsf{GarbIn}, \mathsf{GarbOut}$ and $\mathsf{Translate}$ algorithms and the computational cost of these algorithms do not depend on the circuit size. Finally, note that the interaction between P_1 and P_2 is minimal and only takes place in the context of coin tossing (Step 1 of the protocol). In addition, since this interaction is independent of the function f and of the parties' inputs x and y, it can be performed off-line and for many instances of the protocol at once. The server and P_1 will do work that is linear in the size of the circuit. If the server-aided protocol is run multiple times for the same or different functionalities, we can reduce the online work of P_1 by performing the garbling of multiple circuits (for the same or different functions) in the offline phase. The online work for P_1 would then be similar to P_2 and only linear to the size of his input and output.

Extending to multiple parties. Our protocol can be easily extended to multiple parties as follows. All the parties, except for the server, begin by executing a coin-tossing protocol which results in them sharing a set of coins. The coins are then used by one of parties to garble the circuit and all the parties to garble their inputs. The garbled circuit and inputs are then sent to the server who evaluates the circuit and returns the garbled outputs to the parties.

It is not difficult to show that this extended protocol is secure when either: (1) the server is malicious and the other parties are semi-honest; or (2) the server is isolated and semi-honest, the garbler is semi-honest and the remaining parties are malicious.

7 Protecting Against Deviating Circuit Garblers

The security of the FKN protocol critically relies on the circuit garbler being honest (or semi-honest) and breaks down completely if this is not the case. To add robustness, we augment the protocol to handle the case where player P_1 deviates from the protocol but is non-cooperative (with respect to S).

Using the semi-honest server for verification. If we assume that the server is semi-honest, there is a simple strategy for protecting against deviating but non-cooperative circuit garblers: similar to the protocol of the previous section, P_1 and P_2 run the coin-tossing protocol and as before P_1 uses the retrieved randomness to generate a garbled circuit. This time, however, P_2 also generates the garbled circuit (using the same randomness) and sends his version of the circuit to the server. The server then verifies that the two garbled circuits he receives are the same, and if so, proceeds with the rest of the computation. Note that as long as one of the players is semi-honest, dishonestly garbled circuits are detected by the honest server. This approach yields a server-aided protocol with a significantly better *average efficiency gain* (see Section 5 for an overview of different notions of efficiency gain) compared to the standard two-party Yao protocol with security against malicious adversaries, since it avoids the cut-and-choose steps.

Our goal, however, is to design a protocol that maintains the benefits of the previous protocol (i.e., security against a non-cooperative server) and simultaneously provides protection in cases where the circuit garbler P_1 is non-cooperative with respect to S.

To protect against a non-cooperative P_1, we use the existing cut-and-choose techniques for Yao's garbled circuit protocol (e.g. see [MF06, LP07]). Note that here the cut-and-choose step cannot take place between P_1 and P_2 since that would significantly increase the work of P_2 and, to a large extent, diminish our ultimate goal of gaining efficiency. Hence, we construct a cut-and-choose protocol between P_1 and an *untrusted* server instead. Note that some subtleties arise that are specific to our server-aided setting and require modifications to the way the cut-and-choose steps are performed.

The computational cost of the resulting protocol increases by a factor of λ (the number of circuits) for the players and the server. However, the new protocol inherits the two important efficiency advantages of the previous one, i.e., the computation still only consists of secret-key operations and P_2's computation is only linear in the size of his input and output and, in particular, is independent of the circuit size.

Standard cut-and-choose. In a standard cut-and-choose, P_1 sends multiple copies of the garbled circuit to S. S then asks P_1 to open the secrets related to a subset of those circuits. S verifies the correctness of the opened circuits, evaluates the remaining circuits (called evaluation circuits) and outputs the majority result as the final output. Note that it is essential to compute the majority output and not abort immediately after seeing an inconsistent output. As discussed in previous work (e.g. see [MF06, LP07]), abort in this situation would reveal additional information to a deviating P_1 about P_2's input. Furthermore, to enable P_2 to compute the correct majority output, additional care is needed to make sure P_1 provides the same input to most of the circuits evaluated by S. Avoiding this extra equality-check would undermine both the correctness (by returning the wrong answer to P_2), and the privacy (by allowing P_1 to learn a different function of the inputs) of the protocol. In [LP07] and [MF06], additional consistency-checking mechanisms are added to the cut-and-choose step in order to guarantee the equality of inputs to most of the circuits. Since the techniques from the two papers are similar, and both would work for our server-aided construction, we give a general description of the mechanism that includes both approaches as a special case. More precisely, in addition to the garbled circuits a collection of *input-equality widgets* are also computed by P_1 and sent along with the garbled circuits to S. During the opening phase of the cut-and-choose, a subset of these input-checking widgets are also opened and verified. This step ensures that unless the majority of P_1's garbled inputs (to the evaluation circuits) are the same, his deviation from the protocol will be detected with high probability during the opening phase.

7.1 What goes wrong in the server-aided setting?

We need to address three issues with the above cut-and-choose strategy when it is applied in the server-aided model.

1. First, since P_1 and P_2 independently compute and send to S their garbled inputs for the evaluation circuits, and since we still want to protect against a deviating P_2, P_1 needs to generate the *input-checking widgets* for P_2's input wires as well. This modification could potentially introduce a new security problem. Particularly, for one pair of circuits, P_1 can issue a bad equality-check for a specific bit value of P_2's input wires (e.g. 0) and a correct equality-check for the other bit value (e.g. 1). In the case that the pair of circuits are chosen for evaluation (which happens with non-negligible probability), if S aborts, P_1 concludes that P_2's input is 0 and if he does not, he concludes that P_2's input bit is a 1. However, this problem is easy to address. In fact, S need not abort if the input keys for two circuit he is evaluating do not pass the checks. He can simply evaluate the subset of remaining circuits that pass the checks and assign "invalid" outputs to the rest (without aborting). The final majority output is also computed, by taking this "invalid" outputs into consideration. In the rest of this section, whenever we talk about retrieving the majority output, we refer to this approach.

2. Second, a more subtle issue arises when the server tries to send the majority output as the final result to the parties. In the standard cut-and-choose, the circuit evaluator learns the actual outputs to all the evaluation circuits and therefore can easily determine the majority. In the server-aided setting, however, we do not allow the server to learn the actual output values. S only learns the garbled outputs and therefore cannot determine the majority output on his own.

   First attempt. Initially, one might try to resolve this issue by sending the computed garbled outputs to P_1 and P_2 and requiring them to compute the majority output on their own (note that P_1 and P_2 know the translation table). Unfortunately, this solution compromises the security of the protocol. For instance, this allows a deviating P_1 to learn "too much" information, by sending only a constant number of bad circuits (e.g. circuits that compute a function other than the agreed one), and learning multiple functions of P_2's input with a non-negligible probability of not getting caught.

   Second attempt. An alternative solution is not to reveal to the server the mapping of output keys to their actual bit values, but to map them to two random values k_0 and k_1 corresponding to 0 and 1, respectively. While this would prevent the server from learning the output of computation, it makes the protocol insecure in the scenario where the server is the dishonest party. In particular, this allows a deviating server to return either k_0 or k_1 as the correct output of computation even if it is not the right output.

   As mentioned earlier, we are interested in a solution that allows a semi-honest server to compute the majority output without learning the output itself (output privacy) and at the same time can provide a guarantee that a deviating server would not be able to modify the result of the computation.

Our oblivious cut-and-choose method. The high level idea behind our solution is as follows. After the opening phase, $\lambda/2$ unopened circuits remain. For each output wire, P_1 and P_2 generate two polynomials g_0 and g_1 of degree $\lambda/4$ over the finite field $GF(2^\lambda)$. g_0 and g_1 are chosen uniformly from the space of *permutation polynomials* of a special form which we will discuss shortly. Let $k_0 = g_0(0)$ and $k_1 = g_1(1)$. k_0 and k_1 are used as keys corresponding to 0 and 1, respectively.

P_1 and P_2 evaluate g_0 at keys corresponding to 0 in the $\lambda/2$ evaluation circuits, and g_1 at keys corresponding to 1. These evaluations along with two ciphertexts $c_0 = E_{k_0}(0^\lambda)$ and $c_1 = E_{k_1}(0^\lambda)$ are sent to S and the same process is repeated for all output wires.

These evaluations can be seen as Reed-Solomon encoding of the keys k_0 and k_1. S then evaluates the remaining $\lambda/2$ circuits and uses a Reed-Solomon decoding algorithm to recover one of k_0, or k_1 (S obliviously decodes both but only one of the keys decrypts the corresponding c_i to 0^λ). Let b be the correct output for the wire we are discussing. In case of a deviating P_1, the error correcting

property of the Reed-Solomon codes ensures that as long as a "sufficiently large" fraction of the garbled outputs are correct, the decoding algorithm correctly decodes the correct key k_b (and hence this fulfils the servers' search for the majority output). On the other hand, in the adversarial scenario where the server is dishonest, we argue that he does not learn anything about k_{1-b}. This is exactly where we need g_0 and g_1 to be randomly chosen permutation polynomials (Dickson polynomials) of a special form. Intuitively, if g_{1-b} is a permutation over the finite field, the knowledge of its evaluations at uniformly random evaluation points does not reveal any information about the polynomial itself (and specifically its constant coefficient) to S since all the permutation polynomials of the same form as g_{1-b} can be evaluated to the same values given the right evaluation points, and hence are equally likely to have been chosen. This intuition is formalized in the proof of security for the case when the server is deviating.

How to sample g_0 and g_1. A Dickson polynomial of degree n denoted by $D_n(x, \alpha)$ is given by

$$D_n(x,\alpha) = \sum_{i=0}^{\lfloor n/2 \rfloor} \frac{n}{n-i}\binom{n-i}{i}(-\alpha)^i x^{n-2i}$$

Dickson polynomials have the nice property of acting as permutations of finite fields. More accurately, we have the following lemma about them.

Lemma 7.1. *The Dickson polynomial $D_n(x, \alpha)$ is a permutation polynomial for a finite field of size q if n is coprime to $q^2 - 1$.*

Hence, in order for g_0 and g_1 to be permutation polynomials, we require that $\lambda/4$ is coprime to $2^{\lambda+1} - 1$. In addition, for g_0 and g_1 to have constant coefficients, $\lambda/4$ needs to be even (due to the way Dickson polynomials are defined). Finally, we require that if α is chosen uniformly at random in $GF(2^\lambda)$, the constant coefficient of $D_{\lambda/4}(x, \alpha)$ is a uniformly random element of the field too. The reason for this last requirement is so that we can use the constant coefficients as random keys k_0 and k_1 in the protocol. Since the constant coefficient of $D_{\lambda/4}(x, \alpha)$, for an even value of $\lambda/4$, is of the form $2\alpha^{\lambda/8}$, we essentially need that $\alpha^{\lambda/8}$ be a permutation over the field as well. Once again, according to Lemma 7.1 this is case if $\lambda/8$ is coprime to $2^{\lambda+1} - 1$ since $D_{\lambda/8}(x, 0) = x^{\lambda/8}$ is itself a Dickson polynomial.

To summarize, we sample g_0 and g_1 by choosing λ such that $\lambda/4$ is even and coprime to $2^{\lambda+1} - 1$ (this already implies that $\lambda/8$ is coprime to $2^{\lambda+1} - 1$); generating two random elements $\alpha_0, \alpha_1 \in GF(2^\lambda)$; and letting $g_0 = D_{\lambda/4}(x, \alpha_0)$ and $g_1 = D_{\lambda/4}(x, \alpha_1)$.

Finally, we note that choosing a λ that satisfies the above properties is fairly easy. For example, any $\lambda = 8p$ where p is a prime number is coprime to $2^{\lambda+1} - 1$ and hence can be used in our protocol. To obtain a λ close to 80, one could let $p = 11$. For a λ close to 128, once could let $p = 17$, and so on.

3. A third issue in our setting is that a deviating P_1 can cheat by agreeing on a seed r_i with P_2 but using a different seed r'_i to generate the garbled circuit which he sends to S. More specifically, consider the two garbled circuits $\mathrm{G}_i(C) \leftarrow \mathsf{GarbCircuit}(C; r_i)$ and $\mathrm{G}'_i(C) \leftarrow \mathsf{GarbCircuit}(C; r'_i)$ generated by the two different seeds. A dishonest P_1 can potentially choose the seed r'_i such that for a particular input wire of P_2, the key k corresponds to a 0 in $\mathrm{G}_i(C)$ but corresponds to a 1 in $\mathrm{G}'_i(C)$. In other words, P_1 can flip P_2's input bit without P_2 or S detecting it. This issue does not arise in the standard two-party variant of Yao's protocol against malicious adversaries, due to the existence of the OTs. In the two-party case, P_1 and P_2 engage in a series of OTs (before the cut-and-choose step) as a result of which P_2 learns his garbled inputs for all circuits. In the opening phase, P_2 can verify that the OTs

were performed honestly for the opened circuits, and hence gain confidence about the correctness of his garbled inputs in the majority of the unopened circuits too. Since we no longer invoke OTs in our server-aided protocols, we need a different mechanism for resolving this issue.

First attempt. One may try to solve this problem by making S ask both P_1 and P_2 to send him the seeds for the opened circuits, and verify that the two sets of seeds are equal. However, this creates a new vulnerability for the case when S is dishonest since he could ask for two different subset of circuits to be opened by each of P_1 and P_2. This allows S to learn either P_1 or P_2's input values.

The correct solution is to have S send the set of seeds he receives from P_1, to P_2 who can verify that they are equal to his own seeds and abort the protocol, otherwise (see step 5 of the protocol).

In Theorem 7.2 below we show that our protocol is secure against the following adversary structure:

$$\mathsf{ADV}_2 = \mathsf{ADV}_1 \bigcup \left\{ \left(\mathcal{A}_S[h], \mathcal{A}_1[m], \mathcal{A}_2[h] \right), \left(\mathcal{A}_S[sh], \mathcal{A}_1[nc_S], \mathcal{A}_2[sh] \right) \right\}.$$

Theorem 7.2. *The protocol in Figure 2 securely computes the function f in the $\mathcal{F}_{\mathsf{ct}}$-hybrid model against the adversary structure ADV_2.*

Proof. The proof for the first adversarial model of ADV_1 (where all three parties are semi-honest) is almost identical to the proof for the same model in Theorem 6.2 and hence is omitted here. Next, we prove security against the remaining adversarial scenarios in the adversary structure.

Claim. The protocol $(\mathcal{A}_S[m], \mathcal{A}_1[h], \mathcal{A}_2[h])$-securely computes f in the $\mathcal{F}_{\mathsf{ct}}$-hybrid model.

Consider the simulator $\mathbf{Sim}_S$ that simulates $\mathcal{A}_S$ as follows. It chooses coins $r_1, \ldots, r_\lambda, r'$ and computes $\mathrm{G}_i(C) \leftarrow \mathsf{GarbCircuit}(C; r_i)$ for $1 \leq i \leq \lambda$. $\mathbf{Sim}_S$ then sends $\mathrm{G}_i(C)$ for all $1 \leq i \leq \lambda$ to $\mathcal{A}_S$. $\mathcal{A}_S$ returns a subset T of size $\lambda/2$. For each $i \in T$, $\mathbf{Sim}_S$ returns r_i to $\mathcal{A}_S$.

$\mathbf{Sim}_S$ then chooses random inputs x' for P_1 and y' for P_2. He then sends the garbled input labels $\mathrm{G}_i(x') \leftarrow \mathsf{GarbIn}(C, 1, x'; r_i)$ and $\mathrm{G}_i(y') \leftarrow \mathsf{GarbIn}(C, 2, y'; r_i)$ to $\mathcal{A}_S$ for all $i \in S - T$. Denote by ℓ_0, the number of output wires in each circuit. For each $1 \leq i \leq \ell_o$, $\mathbf{Sim}_S$ use the seed r' to generate two random Dickson polynomials g_0^i and g_1^i of degree $\lambda/4$, and evaluates them at the $\lambda/2$ keys for output wire i. Denote the constant coefficients of g_0^i and g_1^i by k_0^i and k_1^i respectively. $\mathbf{Sim}_S$ then sends the evaluations along with the ciphertexts $c_0^i = E_{k_0^i}(0^\lambda)$ and $c_1^i = E_{k_1^i}(0^\lambda)$ on behalf of P_1 and P_2 to $\mathcal{A}_S$. For each $1 \leq i \leq \ell_o$, $\mathbf{Sim}_S$ receives a key k_b^i. If k_b^i is not a valid key, the simulator instructs the trusted party to return $\perp$ to P_1 and P_2. $\mathbf{Sim}_S$ then outputs whatever $\mathcal{A}_S$ does and halts.

The view of $\mathcal{A}_S$ consists of the garbled circuits, the opened seeds, the garbled input values, and evaluations of the Dickson polynomials at the output keys. The garbled input values that correspond to zero and one are indistinguishable for the adversary since he does not know the seed for the unopened circuits (correctness property in definition A.1). Therefore the garbled labels for the real inputs x and y in the real execution and the random values x' and y' in the ideal execution are indistinguishable for $\mathcal{A}_S$. All the other messages $\mathcal{A}_S$ receives are the same things he would see in the real protocol and hence it follows the views of the adversary in the real and the ideal execution are therefore indistinguishable.

The remaining issue is to prove that the outputs of P_1 and P_2 are also indistinguishable in the real and the ideal execution. Note that they receive the correct output, if $\mathcal{A}_S$ computes and returns the result, honestly. Otherwise, in the ideal execution they receive $\perp$ from the trusted party. In the real execution, for $\mathcal{A}_S$ to return an incorrect output key k_{1-b}^i for a specific output wire i, he needs to guess the constant coefficient of the corresponding polynomial g_{1-b}^i. But, the only knowledge $\mathcal{A}_S$ has of this polynomial is the fact that it is a uniformly random Dickson polynomial of degree $\lambda/4$ that evaluates to values $y_1, \ldots, y_{\lambda/2}$ at

Inputs: P_1's input is x, P_2's input is y, and S has no inputs. All parties know the circuit C which computes the functions f. They also agree on an integer λ such that $\lambda/8$ is coprime to $2^{\lambda+1}-1$.
Outputs: P_1 and P_2 learn the output $f(x,y)$.

1. P_1 and P_2 execute a coin tossing protocol to generate $\lambda+1$ sets of coins $r_1,\ldots,r_\lambda$, and r'.
2. P_1 computes λ garbled circuits $(\mathrm{G}_i(C) \leftarrow \mathsf{GarbCircuit}(C;r_i)$ and garbled inputs $\mathrm{G}_i(x) \leftarrow \mathsf{GarbIn}(C,1,x;r)$ for $1 \le i \le \lambda$. λ is chosen such He sends the computed garbled circuits and inputs to S. He also sends the *input-equality widgets* corresponding to those garbled circuits to S.
3. S randomly chooses a subset $T \subset [1\ldots\lambda]$ of size $\lambda/2$ and asks P_1 to send the coins r_i for $i \in T$, and also reveal the secrets or the equality-checkers corresponding to the opened circuits.
4. S verifies that the opened circuits and input-equality widgets were generated correctly. If the verification fails for any circuit, S aborts.
5. S also sends the coins r_i for $i \in T$ to P_2 who checks if the coins are what he agreed on with P_1 in the coin-tossing phase or not. If not, he aborts the protocol.
6. P_1 and P_2 separately send their garbled inputs for the remaining circuits to S. S uses the remaining input-equality widgets to determine the subset of the circuits in S that pass the input-equality checks. In what follows, the evaluation result for those circuits that fail the checks is set to "invalid" by the server (without aborting).
7. At this point, $t=\lambda/2$ circuits remain to be evaluated. Renumber the remaining circuits as $C_1,\ldots,C_t$. Denote by ℓ_o the number of output wires in each circuit, and let $(w^i_{1,0},w^i_{1,1}),\cdots,(w^i_{t,0},w^i_{t,1})$ the keys corresponding to the output wire $1 \le i \le \ell_o$, in the t circuits. For all $1 \le i \le \ell_o$, P_1 and P_2 use the coins r' to independently generate the following
   - Two uniformly random elements $\alpha^i_0,\alpha^i_1 \in GF(2^\lambda)$, and the corresponding Dickson polynomials $g^i_0(x)=D_{\lambda/4}(x,\alpha^i_0)$, and $g^i_1(x)=D_{\lambda/4}(x,\alpha^i_1)$ over $GF(2^\lambda)$. Denote their corresponding constant coefficients by k^i_0 and k^i_1, respectively.
   - P_1 and P_2 also compute two ciphertexts $c^i_0=E_{k^i_0}(0^\lambda)$ and $c^i_1=E_{k^i_1}(0^\lambda)$ using a symmetric-key encryption scheme E.
8. P_1 and P_2 then independently compute $Y^i_0=(g^i_0(w^i_{1,0}),\ldots,g^i_0(w^i_{t,0}))$ and $Y^i_1=(g^i_1(w^i_{1,1}),\ldots,g^i_1(w_{t,1}))$ for $1 \le i \le \ell_o$ and send $((Y^i_0,c^i_0),(Y^i_1,c^i_1))$ to S. Each pair will be sent in a randomly permuted order so that S does not learn which one corresponds to 0 and which one corresponds to 1. The randomness for the permutation will also be derived from the coins r', and hence P_1 and P_2 will both do the same permutations.
9. S checks that the permuted $((Y^i_0,c^i_0),(Y^i_1,c^i_1))$ pairs he receives from P_1 and P_2 are in fact the same. If not, he aborts. Else, S evaluates the remaining t circuits and for each output wire $1 \le i \le \ell_o$, retrieves the keys $X^i_b=(w^i_{1,b},\ldots,w^i_{t,b})$ for a bit $b \in \{0,1\}$ where b is the actual output value for wire i. (With high probability, only a *small* fraction of these keys will be corrupted or "invalid" or else S would catch them in the opening phase).
10. For every $1 \le i \le \ell_o$, S runs the Reed-Solomon decoding algorithm on both pairs (X^i_b,Y^i_0) and (X^i_b,Y^i_1) to recover the decodings d^i_0 and d^i_1. S uses d^i_0 to decrypt c^i_0 and uses d^i_1 to decrypt c^i_1. With high probability, only the decryption with d^i_b returns the message 0^λ. S returns d^i_b for $1 \le i \le \ell_o$ to P_1 and P_2.
11. P_1 and P_2 use their translation tables to separately recover the actual output values.

Figure 2: A server-aided two-party protocol robust against a deviating P_1

$\lambda/2$ uniformly random points that are unknown to him. The fact that these evaluation points are unknown to $\mathcal{A}_S$ is implied by the verifiability property of Yao's garbled circuit (see Definition A.3).

It remains for us to show that seeing $y_1, \ldots, y_{\lambda/2}$ does note leak any information about the underlying polynomial g^i_{1-b} (and particularly its constant coefficient). However, since g^i_{1-b} is a permutation and the evaluation points are uniformly random values from $GF(2^\lambda$, y_i's essentially constitute $\lambda/2$ uniformly random values in $GF(2^\lambda)$, and hence do not contain any information about g^i_{1-b}. Put differently, for any Dickson polynomials of degree $\lambda/4$ $p(x)$, there exist a unique set of evaluation points $x_1, \ldots, x_{\lambda/2}$ such that $p(x_i) = y_i$ for $1 \leq i \leq \lambda/2$. This completes our argument.

□

Claim. The protocol $\big(\mathcal{A}_S[h], \mathcal{A}_1[m], \mathcal{A}_2[h]\big)$-securely computes f in the $\mathcal{F}_{\mathsf{ct}}$-hybrid model.

We construct a simulator $\mathbf{Sim}_1$ for $\mathcal{A}_1$. Before describing the simulation, however, we give a Lemma implied by the security of the cut-and-choose variants of Yao's protocol (we refer the reader to [LP07] where the Lemma is implicitly proved). We point out, however, that in our setting, the role of the honest verifying party is divided between an honest server and an honest P_2, and hence the Lemma goes through since their verification mechanisms combined is equivalent to the verification mechanism performed by the single honest party in the standard two-party case of malicious Yao. In particular, it is essential for both the server to check the correctness of the opened circuits and the input-equality widgets in steps 3 and 4, and for the honest P_2 to check the correctness of the revealed seeds in step 5. This point is also dircetly related to the third issue we explored in the discussion above.

Lemma 7.3 ([LP07])**.** *There exists an expected polynomial time extractor* Ext*, that takes P_1's input x and runs and rewinds $\mathcal{A}_1$. If $\mathcal{A}_1$ creates more than $3/4$ of the evaluation circuits honestly (a total of $3\lambda/8$ circuits), and provides the same input x' for all of those correct circuits, w.h.p.,* Ext *extracts and outputs x'. Else,* Ext *will output $\perp$ with all but negligible probability. Furthermore, the probability of* Ext *outputting $\perp$ when interacting with $\mathcal{A}_1$ is exactly the same as the probability of an honest server outputting $\perp$ in the real protocol.*

The fraction $3/4$ in the above Lemma is adjustable to any constant fraction greater than $1/2$. Changing the constant fraction affects the extractor's probability of error in the Lemma, but that probability still remains negligible in the security parameter. In order to ensure correct decoding, we need a constant fraction of $3/4$ or higher in our proofs.

Our transformation $\mathbf{Sim}_1$ works as follows:

1. it makes a query to $\mathcal{F}_{\mathsf{ct}}$, and $\mathcal{A}_1$ answers back with $r_1, \ldots, r_\lambda$, and r'.
2. it runs the extractor Ext from Lemma 7.3 to either obtain the input x' that $\mathcal{A}_1$ has provided for the majority of the circuits, or to receive the abort signal $\perp$. For the latter, $\mathbf{Sim}_1$ simulates the server aborting and outputs whatever $\mathcal{A}_1$ does.
3. it uses r' to generate the permutation polynomials $g^i_0(x)$ and $g^i_1(x)$ with constant coefficients $k^i_0, k^i_1 \in \{0,1\}^\lambda$, for $1 \leq i \leq \ell_o$.
4. it sends x' to the trusted party and obtains $z = f(x', y)$. For bits b_i of the output z, he selects the corresponding key $k^i_{b_i}$, (for $1 \leq i \leq \ell_o$) and returns them to P_1.

We show that $\mathcal{A}_1$ cannot distinguish his interaction with an honest S and P_2 in the real world from his interaction with $\mathbf{Sim}_1$ in the ideal world. The view of the adversary consists of the garbled circuits he submits and the output he receives. Therefore, it is enough to show that the output that $\mathcal{A}_1$ receives in the real execution is the same as what he receives in the above simulation. Based on the existence of the extractor Ext from Lemma 7.3, we are assured that in case of an abort, the view of $\mathcal{A}_1$ in the simulation

is indistinguihsable from when he interacts with the honest server. Furthermore, in the case that Ext extracts an input x', we know that with high probability 3/4 of the evaluation circuits (i.e., $3\lambda/8$ circuits) constructed by $\mathcal{A}_1$ are correct and use the same input x'. Let $z = f(x', y)$. We now have that for every bit b_i of z, the honest server in the real execution, receives $\lambda/2$ evaluation points for polynomial g_{b_i} of degree $\lambda/4$ and that $3\lambda/8$ of those are correct. In other words, the error in the Reed-Solomon codeword is less than $\lambda/8 < (\lambda/2 - \lambda/4)/2$. Hence, the Reed-Solomon decoding algorithm run by the honest server unambiguously recovers the garbled inputs k_{b_i} for all $1 \leq i \leq \ell_o$. Since the server runs the decoding obliviously on both polynomials, however, we also need to show that for polynomials g_{1-b_i}, the decoding will always fail to return a valid key k_{1-b_i}. However, this is the case since for each polynomial g_{1-b_i} at most $\lambda/8 = \lambda/2 - 3\lambda/8$ of the evaluation points are correct while the degree of the polynomial is $\lambda/4$. Hence, the RS decoding algorithm either fails or returns a key $k \neq k_{1-b_i}$ for all $1 \leq i \leq \ell_o$. Consequently, S will not be able to correctly decrypt c^i_{1-b} to the message 0^λ using the key k. This completes our argument.

□

□

Claim. The protocol $\big(\mathcal{A}_S[h], \mathcal{A}_1[h], \mathcal{A}_2[m]\big)$-securely computes f in the $\mathcal{F}_{\mathsf{ct}}$-hybrid model.

The simulation for this case is very similar to the case of malicious P_1. In fact, note that P_2 can only perform a subset of cheating strategies of P_1, by sending bad or inconsistent garbled inputs (but not garbled circuits). Hence, a simpler variant of the extractor Ext in Lemma 7.3 can be used to simulate $\mathcal{A}_2$ in the ideal world. Given Ext, the remainder of $\mathbf{Sim}_2$'s strategy will be identical to that of $\mathbf{Sim}_1$ described above, and the same analysis goes through.

□

Claim. The protocol $\big(\mathcal{A}_S[sh], \mathcal{A}_1[nc_S], \mathcal{A}_2[sh]\big)$-securely computes f in the $\mathcal{F}_{\mathsf{ct}}$-hybrid model.

The proof of this claim is automatically implied given the proofs of the above claims and Lemma 6.1.

□

■

7.2 Extending to Multiple Parties

We describe, at a high level, how our protocol can be extended to the multi-party setting. All parties engage in the coin-tossing protocol and learn the necessary seeds. Then, the circuit garbler and the server proceed as they would in the the two-party case. All other parties (clients) who need to send their garbled inputs to the server, engage in a MPC protocol between themselves and the server, wherein their inputs is the seeds they hold and their own input keys, and the server's output is the input key for all parties. Note that this MPC protocol needs to be secure against non-cooperative parties (except for one semi-honest party and a semi-honest server), and hence, the naive solution of having each party send their input key to the server would not be sufficient. Nevertheless, due to the particular setting we work in, a very efficient construction for implementing this MPC protocol exists, but we defer a more detailed description of this step to a more complete version.

In addition to sending their garbled inputs, we require all the parties to send the server the evaluations of the polynomials g^0_i and g^1_i at the garbled output values of the garbled circuits used in the protocol. The server then checks that all the parties sent the same evaluations for g^0_i and g^1_i and if so, uses those values for the interpolation of the garbled outputs. Otherwise, the server aborts.

In addition to security against ADV_1, this protocol also provides security when all but one of the parties are non-cooperative with respect to the server and the server and a single party are semi-honest. Furthermore, if we modify the coin tossing protocol to include the server but without providing him with the result of the coin-toss, the protocol can also handle an adversarial structure where all parties (except for the server) are non-cooperative with respect to the server. We note that an adversarial structure where *all* parties (except the server) are non-cooperative is meaningful since this corresponds to cases where each party can deviate from the protocol but does not share any private information or coordinate its behavior with other parties.

8 Server-Aided Computation From Delegated Computation

We describe a general construction for server-aided two-party computation based on any non-interactive delegated computation scheme and any secure two-party computation protocol. The resulting server-aided construction inherits the efficiency of the underlying protocols.

Informally, a delegated computation scheme allows a client to outsource the computation of a circuit C on a private input x to an untrusted worker such that: (1) the client's work is substantially smaller than evaluating $C(x)$ on his own; (2) the worker does not learn any information about the client's input or output; and (3) the worker cannot return an incorrect answer without being detected. The notion of secure delegated computation is closely related to that of verifiable computation [GKR08, GGP10, CKV10], with the additional requirement that the client's input and output remain private.

Our protocol. Our protocol, described in detail in Figure 3, works as follows. Let $\mathsf{Del} = (\mathsf{Gen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ be a secure delegated computation scheme. The parties P_1 and P_2 use Del to outsource the evaluation of f on their combined inputs (i.e., the concatenation of x and y) to the server. However, since P_1 and P_2 do not want to reveal their inputs to each other, they use secure two-party computation to simulate the client in a delegated computation interaction. More precisely, they run the Gen, $\mathsf{ProbGen}$ and Verify algorithms of Del via secure two-party computation. The server S performs the same functionality as that of the worker in the delegated computation interaction. Put differently, we use Del to outsource a two-party computation protocol between P_1 and P_2 to the server S.

Intuitively, our protocol is secure since: (1) the verifiability of Del guarantees that a malicious server cannot change the outcome of the computation; and (2) the privacy property Del guarantees that the server cannot learn any useful information about the parties' inputs. Furthermore, even if a malicious server colludes with either P_1 or P_2, the colluding parties will not learn any information about the remaining party's input since the interaction between P_1 and P_2 is done via a secure two-party computation. We prove this intuition in Theorem 8.1 using the following adversary structure:

$$\begin{aligned}\mathsf{ADV}_4 = \Big\{ &\Big(\mathcal{A}_S\big[sh\big], \mathcal{A}_1\big[sh\big], \mathcal{A}_2\big[sh\big]\Big), \\ &\Big(\mathcal{A}_S\big[m\big], \mathcal{A}_1\big[h\big], \mathcal{A}_2\big[h\big]\Big), \\ &\Big(\mathcal{A}_S\big[nc_1, nc_2\big], \mathcal{A}_1\big[sh\big], \mathcal{A}_2\big[sh\big]\Big)\Big\}.\end{aligned}$$

Theorem 8.1. *If* Del *is secure, then the server-aided two-party protocol described in Figure 3 is secure in the* $\mathcal{F}_{\mathsf{2pc}}$*-hybrid model against the adversary structure* ADV_4*.*

Proof. We sketch the proofs for each item in ADV_4 separately.

Inputs P_1's input is x and P_2's input is y. Server S does not have any input.
Output P_1 and P_2 want to learn the function F on their inputs. Without loss of generality we assume that F takes x and y as one concatenated input, i.e., the parties are computing $F(x||y)$.

1. P_1 and P_2 use the ideal functionality $\mathcal{F}_{\mathsf{2pc}}^{f}$, where f takes as input x and y, computes $(\text{PK}, \text{SK}) \leftarrow \mathsf{Gen}(1^k, F)$ and $(\sigma_{x||y}, \tau_{x||y}) \leftarrow \mathsf{ProbGen}_{\text{SK}}(x||y)$ and outputs

$$\Big(\big(\text{PK}, \sigma_{x||y}, \tau^1_{x||y}, \text{SK}^1\big), \big(\text{PK}, \sigma_{x||y}, \tau^2_{x||y}, \text{SK}^2\big)\Big),$$

   where $\tau_{x||y} = \tau^1_{x||y} \oplus \tau^2_{x||y}$ and $\text{SK} = \text{SK}^1 \oplus \text{SK}^2$. In essence P_1 and P_2 both learn PK and $\sigma_{x||y}$ while they each only learn a random share of $\tau_{x||y}$ and SK.
2. P_1 and P_2 each send $\sigma_{x||y}$ and PK to the server S but keep their shares of $\tau_{x||y}$ to themselves. Server checks to see that the two values received from the parties are the same and aborts otherwise.
3. Server S computes $\mathsf{Compute}_{\text{PK}}(\sigma_{x||y}) \rightarrow \sigma_z$ where $z = F(x||y)$ and sends σ_z to P_1 and P_2.
4. P_1 and P_2 use the ideal functionality $\mathcal{F}_{\mathsf{2pc}}^{g}$ where g takes as input $\tau^1_{x||y}, \tau^2_{x||y}, \text{SK}^1$ and SK^2 and outputs (z, z) where $z = \mathsf{Verify}_{\text{SK}^1 \oplus \text{SK}^2}((\tau^1_{x||y} \oplus \tau^2_{x||y}), \sigma_z)$. P_1 and P_2 output z as their final output.

Figure 3: A server-aided two-party protocol from any delegated computation scheme

Claim. The protocol $\big(\mathcal{A}_S\big[sh\big], \mathcal{A}_1\big[sh\big], \mathcal{A}_2\big[sh\big]\big)$-securely computes F in the $\mathcal{F}_{\mathsf{2pc}}$-hybrid model.

We describe three independent transformations $\mathbf{Sim}_S$, $\mathbf{Sim}_1$ and $\mathbf{Sim}_2$:

- $\mathbf{Sim}_S$ runs $\mathcal{A}_S$. $\mathbf{Sim}_S$ then generates two arbitrary inputs x', y', computes $(\text{PK}, \text{SK}) \leftarrow \mathsf{Gen}(1^k, F)$ and $(\sigma_{x'||y'}, \tau_{x'||y'}) \leftarrow \mathsf{ProbGen}_{\text{SK}}(x'||y')$ and sends $\sigma_{x'||y'}$ to $\mathcal{A}_S$. The privacy property of Del ensures that $\mathcal{A}_S$'s view is indistinguishable from its view in the real-world execution with semi-honest P_1 and P_2 (where they use their real inputs x and y). At some point, $\mathcal{A}_S$ sends the output σ_z. $\mathbf{Sim}_S$ then outputs whatever $\mathcal{A}_S$ does and halts. Since, $\mathcal{A}_S$ is semi-honest in this case, this will be the correct output.

- $\mathbf{Sim}_1$ runs $\mathcal{A}_1$. Note that since we prove the security of the protocol in the $\mathcal{F}_{\mathsf{2pc}}$-hybrid model, $\mathcal{A}_1$ will send his input x to $\mathcal{F}_{\mathsf{2pc}}$. $\mathbf{Sim}_1$ forwards x to the trusted party of the ideal execution and gets back $F(x, y)$. $\mathbf{Sim}_1$ generates an arbitrary input y' for P_2, runs $(\text{PK}, \text{SK}) \leftarrow \mathsf{Gen}(1^k, F)$, and runs $\mathsf{ProbGen}_{\text{SK}}(x||y')$ to compute $\sigma_{x||y'}, \tau_{x||y'}$. He then sends $\sigma_{x||y'}$ and random values $\tau^1_{x||y'}$ and SK^1 to $\mathcal{A}_1$. The privacy property of Del guarantees that in $\mathcal{A}_1$'s view, $\sigma_{x||y'}$ is indistinguishable from $\sigma_{x||y}$ for any y and y'. The same is true for $\tau^1_{x||y'}$ and SK^1 which are simply random shares. It is worth noting that the privacy property of Del requires that $\sigma_{x||y'}$ hide all *partial information* about the encoded input. Therefore, the tuple $(x, \sigma_{x||y'})$ is also indistinguishable form the tuple $(x, \sigma_{x||y})$ for any y. Hence, we safely assume that $\mathcal{A}_1$'s view so far is indistinguishable from his view in the real execution with semi-honest P_2 and S.

  $\mathbf{Sim}_1$ then computes $\sigma_{z'} \leftarrow \mathsf{Compute}_{\text{PK}}(\sigma_{x||y'})$ and sends $\sigma_{z'}$ to $\mathcal{A}_1$ on behalf of the server. Note that for the same reason as above and due to the privacy property of Del, $\mathcal{A}_1$ cannot distinguish $\sigma_{z'}$ from σ_z. $\mathcal{A}_1$ eventually sends SK^1 and $\tau^1_{x||y}$ as his input to the trusted party of the $\mathcal{F}_{\mathsf{2pc}}$-functionality for the second run of MPC that runs the Verify function. Since $\mathcal{A}_2$ is semi-honest, $\mathbf{Sim}_1$ simply returns the value $F(x, y)$ that he received from the trusted party of the ideal-world execution to $\mathcal{A}_1$. $\mathbf{Sim}_1$ then outputs whatever $\mathcal{A}_1$ does and halts.

- $\mathbf{Sim}_2$'s strategy is identical to $\mathbf{Sim}_1$'s since their roles in the protocol are symmetric.

□

Claim. The protocol $\left(\mathcal{A}_S[m], \mathcal{A}_1[h], \mathcal{A}_2[h]\right)$-securely computes F in the $\mathcal{F}_{\mathsf{2pc}}$-hybrid model.

We describe a transformation $\mathbf{Sim}_S$ for the adversary $\mathcal{A}_S$. Note that since S does not have any inputs to the protocol, there is no need for input extraction during the simulation. $\mathbf{Sim}_S$ only needs to simulate $\mathcal{A}_S$'s view correctly and make sure that in the case of an abort, or other types of cheating by $\mathcal{A}_S$, P_1 and P_2's output in the ideal execution is an abort as well.

$\mathbf{Sim}_S$ runs $\mathcal{A}_S$. $\mathbf{Sim}_S$ then generates two arbitrary inputs x', y', computes $(\text{PK}, \text{SK}) \leftarrow \mathsf{Gen}(1^k, F)$ and $(\sigma_{x'||y'}, \tau_{x'||y'}) \leftarrow \mathsf{ProbGen}_{\text{SK}}(x'||y')$ and sends $\sigma_{x'||y'}$ to $\mathcal{A}_S$. The privacy property of Del ensures that $\mathcal{A}_S$'s view is indistinguishable from his view in the real execution with honest P_1 and P_2 (where they use their real inputs). At some point, $\mathcal{A}_S$ will either abort or send the output $\sigma_{z'}$ to the two parties. $\mathbf{Sim}_S$ computes $z' \leftarrow \mathsf{Verify}_{\text{SK}}(\tau_{x'||y'}, \sigma_{z'})$. If $z' = \perp$, $\mathbf{Sim}_S$ sends an *abort* message to the trusted party and simulates P_1 and P_2 aborting. $\mathbf{Sim}_S$ then outputs whatever $\mathcal{A}_S$ does and halts.

Note that the verifiability property of Del (see definition B.2) ensures that if $z' \neq F(x', y')$ then $z' = \perp$ with high probability. Also note that the probability that $z' = \perp$ in the simulation with inputs x' and y' is (all but negligibly) close to the same probability for any other inputs including inputs x and y of P_1 and P_2 in the real execution. If this was not the case, once again we could use $\mathcal{A}_S$ to break the privacy property of Del. These two facts combined demonstrate that the joint distribution of outputs of $\mathcal{A}_S$, P_1 and P_2 in the real execution are computationally indistinguishable from those of $\mathbf{Sim}_S$, $\mathbf{Sim}_1$ and $\mathbf{Sim}_2$ in the ideal execution.

□

Claim. The protocol $\left(\mathcal{A}_S[nc_1, nc_2], \mathcal{A}_1[sh], \mathcal{A}_2[sh]\right)$-securely computes F in the $\mathcal{F}_{\mathsf{2pc}}$-hybrid model.

The proof of this claim follows from the previous two claims and Lemma 6.1.

■

Efficiency. Note that in the above protocol P_1 and P_2 only simulate the client in the delegated computation interaction and not the server. Using general-purpose protocols such as Yao's protocol [Yao82], the computation performed by P_1 and P_2 will be linear in that of client in the delegated computation interaction. Given the efficiency properties of delegated computation, this will be significantly lower than P_1 and P_2 running their own secure two-party computation protocol to compute F on their private inputs. The server's computation is identical to that of the worker in the delegated computation interaction.

We know of two non-interactive delegated computation schemes in the literature. The first is the construction of [GGP10] based on Yao's garbled circuits and fully homomorphic encryption. When instantiated using their protocol, P_1 and P_2 would need perform $O(k \cdot |C|)$ computation in the preprocessing stage, where C is the circuit that computes the function F and k is the security parameter. In the online stage however, the work of P_1 and P_2 is $O(n + m)$ where n is the size of their inputs and m the size of their outputs. Hence, the amortized complexity of the work by the players is $O(n+m)$. The server's work will be $O(|C|)$.

The second instantiation is based on the construction of [CKV10] which uses non-interactive proofs with soundness amplification, as well as a fully homomorphic encryption scheme. When instantiated using their protocol, the offline cost of computation for P_1 and P_2 would be $\mathsf{poly}(k, |C|)$ while the online and hence the amortized cost is $\mathsf{poly}(k, \log(|C|))$. The servers computation is $\mathsf{poly}(k, |C|)$. The main advantage of the latter instantiation is that the public key of the scheme will be significantly smaller.

9 Private Set Intersection in the Server-Aided Model

The setting for the problem of set intersection includes two parties that have private input sets and wish to compute the intersection of the elements in their sets. This problem has numerous applications in practice and has been considered in a series of works [FNP04, HL08, KS05, JL09, DSMRY09, CKT10, HN10, DSMRY11]. These papers offer protocols addressing various adversarial models under different assumptions with a range of efficiency characteristics. In the semi-honest adversarial model, the protocols for set intersection have linear complexity in the size of the inputs. The goal of many of these works is to approach this complexity in stronger adversarial models. For instance, Cristofaro et al. [CKT10] achieve linear complexity in the malicious case in the random oracle model. Jarecki et al. [JL09] propose a protocol with linear complexity secure in the standard model in the presence of malicious parties, based on the Decisional q-Diffie-Hellman Inversion assumption (in the CRS model), where a safe RSA modulus is generated by a trusted third party, and the input domain is of size polynomial in the security parameter.

We consider the problem of set intersection in the setting of outsourced computation where we would like to enable the computationally powerful server to execute the majority of the work involved comparing the values in the two input sets. Essentially we are interested in a solution where each of the two parties performs work that is linear in the size of his/her input set to preprocess the data in their sets and send the results to the server who will compute the final intersection result.

A simple solution for the case when all parties are semi-honest. In the case of a semi-honest server we can obtain such a protocol as follows: the two input parties agree on a PRF key, and submit to the server the result of the evaluation of the PRF under this key on each of the points in their input sets. Subsequently the server computes the set intersection of the two sets of PRF values he received, and sends the output to the two parties, who can map the PRF values back to the real input points. As long as the server follows the protocol honestly the two parties will receive the correct output without being able to learn anything about their private data due to the security guarantees of the PRF.

Protecting against a malicious server. The above protocol fails to guarantee correctness of the output in the case of a malicious server who can deviate from the prescribed protocol since he can return an arbitrary result without the parties being able to detect this. We adopt the following technique in order to enable the parties to detect misbehavior on the server's side: each party computes t copies of each of his inputs of the form $x|i$ for $1 \leq i \leq t$, and submits the PRF evaluations on the resulting values in a randomly permuted order. The server then computes the set intersection based on these PRF values and returns the answer. Now, we require that the set intersection contains all t copies for each element in the intersection. If it does not, then the parties will detect misbehavior on server's side and will abort. Thus in order to cheat without being detected, the server will need to guess what values correspond to the copies of the same element. The probability for this is negligible except in the following two cases (1) the server returns empty intersection (does not need to return any value) or (2) claims to each party that all elements from his/her input set are in the intersection (returns all PRF values provided by that party). To address these last issues we need to guarantee that the set intersection is neither empty nor contains all submitted elements. We achieve this in the following way: the parties agree on three elements d, e_1 and e_2 outside the range of possible input values. Then, the first party adds d and e_1 to his/her input set and the second party adds d and e_2 to his/her input set. Now the set intersection has to be non-empty since d will be in it, and at the same time cannot consist of all submitted input elements for either party since both e_1 and e_2 are not in the intersection. The protocol in Figure 4 presents the details of the the approach that we just outlined. We also note that for the purposes of the simulation, we need to use a pseudorandom permutation rather than any pseudorandom function.

Let m and n be the sizes of the inputs sets for parties P_1 and P_2 with elements in the domain R. Let F be a pseudo-random permutation. Let t be a security parameter.

Inputs: P_1 has input set X; P_2 has input set Y

Outputs: P_1 and P_2 receive $X \bigcap Y$

Protocol:

1. P_1 and P_2 run a coin tossing protocol to choose a PRP key K.
2. P_1 and P_2 choose three elements $d, e_1, e_2 \notin R$. P_1 adds d and e_1 to his set X. P_2 adds d and e_2 to his set Y.
3. For each $x_i \in X$, P_1 computes $a_{i,j} = F_K(x_i|j)$ for $1 \leq j \leq t$. P_1 sends the set $A = \{a_{i,j}\}_{1 \leq i \leq m, 1 \leq j \leq t}$ in a randomly permuted order to S.
4. For each $y_i \in Y$, P_2 computes $b_{i,j} = F_K(y_i|j)$ for $1 \leq j \leq t$. P_2 sends the set $B = \{b_{i,j}\}_{1 \leq i \leq n, 1 \leq j \leq t}$ in a randomly permuted order to S.
5. S computes the set $A \bigcap B$ and sends it to P_1 and P_2.
6. P_1 checks that the PRP values corresponding to d are present in $A \bigcap B$ and those corresponding to e_1 are not. He also checks that if $F_K(x_i|j) \in A \bigcap B$ for some $j \in [1, t]$, then $F_K(x_i|j) \in A \bigcap B$ for all $j \in [1, t]$. If either of these checks fails, P_1 aborts the protocol. P_2 runs a similar check.
7. Using K, P_1 and P_2 recover the values in $X \cap Y$.

Figure 4: Security against malicious server

Theorem 9.1. *The protocol in Figure 4 securely computes the* 2*-party set intersection functionality in the* $\mathcal{F}_{\mathsf{ct}}$*-hybrid model for the adversary structure* ADV_5 *defined as follows:*

$$\mathsf{ADV}_5 = \left\{ \left(\mathcal{A}_S[sh], \mathcal{A}_1[sh], \mathcal{A}_2[sh] \right), \left(\mathcal{A}_S[m], \mathcal{A}_1[h], \mathcal{A}_2[h] \right), \left(\mathcal{A}_S[nc_1, nc_2], \mathcal{A}_1[sh], \mathcal{A}_2[sh] \right) \right\}.$$

Proof. We consider the different adversarial models in the following claims.

Claim. The protocol $\left(\mathcal{A}_S[sh], \mathcal{A}_1[sh], \mathcal{A}_2[sh]\right)$-securely computes the 2-party set intersection functionality in the $\mathcal{F}_{\mathsf{ct}}$-hybrid model.

We describe three independent transformations $\mathbf{Sim}_S$, $\mathbf{Sim}_1$ and $\mathbf{Sim}_2$. The simulator $\mathbf{Sim}_1$ simulates $\mathcal{A}_1$ as follows:

1. $\mathbf{Sim}_1$ queries $\mathcal{F}_{\mathsf{ct}}$ to receive the common random string r used to derive K_1, K_2 and d, e_1, e_2 and answers $\mathcal{A}_1$'s queries to $\mathcal{F}_{\mathsf{ct}}$ correspondingly.
2. $\mathbf{Sim}_1$ calls the trusted party submitting the input it has for the semi-honest $\mathcal{A}_1$ and obtains the output for $\mathcal{A}_1$
3. The simulator computes the corresponding PRP values for the output and sends those to $\mathcal{A}_1$.

We construct a simulator $\mathbf{Sim}_2$ that simulates $\mathcal{A}_2$ analogously to $\mathbf{Sim}_1$. The simulator $\mathbf{Sim}_S$ that simulates $\mathcal{A}_S$ as follows:

1. $\mathbf{Sim}_S$ generates two random sets X and Y of size m and n.

2. $\mathbf{Sim}_S$ chooses K_1, K_2 and d, e_1, e_2, computes honestly the PRP values for X and Y and submits them to $\mathcal{A}_S$.

The indistinguishability of the simulated and the real execution views follows easily from the pseudo-random properties of the PRP.

□

Claim. The protocol $\big(\mathcal{A}_S[m], \mathcal{A}_1[h], \mathcal{A}_2[h]\big)$-securely computes the 2-party set intersection functionality in the $\mathcal{F}_{\mathsf{ct}}$-hybrid model.

We construct a simulator $\mathbf{Sim}_S$ that simulates the adversary $\mathcal{A}_S$ as follows:

1. $\mathbf{Sim}_S$ generates two random sets X and Y of size m and n.
2. $\mathbf{Sim}_S$ chooses s_1, s_2 and d, e_1, e_2, computes honestly the PRP values for X and Y and submits them to $\mathcal{A}_S$.
3. $\mathbf{Sim}_S$ receives the output computed by $\mathcal{A}_S$. If the returned set is not the correct set of intersection PRP vales, the simulator sends an abort message to the trusted party and to $\mathcal{A}_S$.

The views of the adversary $\mathcal{A}_S$ in the real and the ideal execution are indistinguishable because of the properties of the PRP function and the fact that P_1 and P_2 are honest. In the ideal execution P_1 and P_2 receive as output the set intersection of their inputs if and only if $\mathcal{A}_S$ has computed it correctly (i.e., $\mathbf{Sim}_S$ has not submitted abort to the trusted party). Thus we need to show that in the real execution the probability that the parties will not abort, when the set returned by $\mathcal{A}_S$ is not the correct result, is negligible. A misbehavior of $\mathcal{A}_S$ will not be detected, if the intersection set that he returns contains all PRP values for the element d, does not contain any of the PRP values for e_1 and e_2, and for every PRP value in the returned set all the other $t-1$ PRP values that correspond to the respective element are also in the claimed intersection set. Let r be size of the set intersection. The probability that $\mathcal{A}_S$ removes $k \leq r$ values from the set intersection without being detected is (i.e., guesses the PRP values that correspond to the element d and then guesses kt PRP values that correspond to k input elements):

$$\binom{(r+1)t}{t}^{-1}\binom{r}{k}\binom{rt}{kt}^{-1}.$$

The probability that $\mathcal{A}_S$ adds $s \leq m-r$ ($s \leq n-r$) values from the set intersection returned to P_1 (P_2) without being detected is (i.e., guesses the t PRP values corresponding to e_1 (e_2) and then guesses st PRP values corresponding to s elements):

$$\binom{(n-r+1)t}{t}^{-1}\binom{(n-r)}{s}\binom{(n-r)t}{st}^{-1}.$$

The value $\binom{(r+1)t}{t}^{-1}$ is maximized when $r=1$ (if $r=0$, $\mathcal{A}_S$ cannot remove intersection values). Therefore,

$$\binom{(r+1)t}{t}^{-1} \leq \binom{2t}{t}^{-1} = \frac{t!t!}{(2t)!} = \frac{1\cdots t}{(t+1)\cdots 2t} < \frac{1}{2^t}.$$

Since $\binom{r}{k}\binom{rt}{kt}^{-1} < 1$, it follows that the probability that $\mathcal{A}_S$ removes any values from the set intersection without being detected is negligible. Similarly we get that the probability of that $\mathcal{A}_S$ adds any values from the set intersection without being detected is also negligible. Therefore, the probability that the set intersection that a party accepts as answer is not the correct result is negligible.

□

Claim. The protocol $\left(\mathcal{A}_S[nc_1, nc_2], \mathcal{A}_1[sh], \mathcal{A}_2[sh]\right)$-securely computes the 2-party set intersection functionality in the $\mathcal{F}_{\mathsf{ct}}$-hybrid model.

The proof of the claim follows from the above two claims and Lemma 6.1.

□

■

Protecting against malicious parties. While the above protocol allows the two parties to detect a malicious server, it introduces a way for a malicious party to submit incorrectly processed input that would cause the other party to abort after receiving an invalid set intersection result, while the misbehaving party will learns the real output. Furthermore, the fact that the honest party aborts can itself leak additional information about his inputs. In order to enable detection of misbehavior on the side of either party, we augment the protocol again. If we did not want to provide privacy of the input data from the server but still wanted to guarantee the correctness of the output result, we could solve this problem as follows: the server commits to the intersection set that he computes from the PRF values, then the parties reveal the key for the PRF to him so that he can verify the correctness of the submitted input sets and notify the two parties while not being able to change the computed output because of the commitment. In order to maintain the privacy for the input sets we can introduce an additional layer of PRF invocation where the first layer will account for the privacy guarantee and the second layer will allow for detection of the correctness of the input sets submitted by each party. We provide the details of the construction in Figure 5.

Theorem 9.2. *The protocol in Figure 4 securely computes the* 2*-party set intersection functionality in the* $\mathcal{F}_{\mathsf{ct}}$*-hybrid model for the adversary structure* ADV_6 *defined as follows:*

$$\mathsf{ADV}_6 = \mathsf{ADV}_5 \bigcup \Big\{ \Big(\mathcal{A}_S[h], \mathcal{A}_1[m], \mathcal{A}_2[h]\Big), \Big(\mathcal{A}_S[sh], \mathcal{A}_1[nc_S], \mathcal{A}_2[sh]\Big), \Big(\mathcal{A}_S[h], \mathcal{A}_1[h], \mathcal{A}_2[m]\Big), \Big(\mathcal{A}_S[sh], \mathcal{A}_1[sh], \mathcal{A}_2[nc_S, nc_1]\Big) \Big\}.$$

Proof. We consider only the cases in the adversarial structure that were not covered in the proof of Theorem 9.1.

Claim. The protocol $\left(\mathcal{A}_S[h], \mathcal{A}_1[m], \mathcal{A}_2[h]\right)$-securely computes the 2-party set intersection functionality in the $\mathcal{F}_{\mathsf{ct}}$-hybrid model.

We construct a simulator $\mathbf{Sim}_1$ that simulates the adversary $\mathcal{A}_1$ as follows:

1. $\mathbf{Sim}_1$ queries $\mathcal{F}_{\mathsf{ct}}$ to receive the common random string r used to derive K_1, K_2 and d, e_1, e_2 and answers $\mathcal{A}_1$'s queries to $\mathcal{F}_{\mathsf{ct}}$ correspondingly.

2. $\mathbf{Sim}_1$ receives from $\mathcal{A}_1$ the PRP values that he submits, and uses K_1 and K_2 to extract the inputs. If he fails to extract these values, $\mathbf{Sim}_1$ submits abort to the TP. Otherwise, $\mathbf{Sim}_1$ submits the extracted values to the trusted party and receives back the set intersection.

3. The simulator verifies that $\mathcal{A}_1$ has submitted exactly t PRP values for each of his inputs. If the verification fails, he instructs the TP to send abort to the P_2.

Let m and n be the sizes of the inputs sets for parties P_1 and P_2 with elements in the domain R. Let G and F be pseudo-random permutation. Let t be a security parameter. Let *com* be a commitment scheme.

Inputs: P_1 has input set X; P_2 has input set Y

Outputs: P_1 and P_2 receive $X \bigcap Y$

Protocol:

1. P_1 and P_2 run a coin tossing protocol to choose two key K_1 and K_2.
2. P_1 and P_2 choose three elements $d, e_1, e_2 \notin R$. P_1 adds d and e_1 to his set X. P_2 adds d and e_2 to his set Y.
3. P_1 computes $X' = \{G_{K_1}(x) \mid x \in X\}$.
4. P_1 computes $Y' = \{G_{K_1}(y) \mid y \in Y\}$.
5. For each $x'_i \in X'$, $1 \leq i \leq m$ P_1 computes $a_{i,j} = F_{K_2}(x_i|j)$ for $1 \leq j \leq t$. P_1 sends the set $A = \{a_{i,j}\}$ to S.
6. For each $y'_i \in Y'$, $1 \leq i \leq n$ P_2 computes $b_{i,j} = F_{K_2}(y_i|j)$ for $1 \leq j \leq t$. P_2 sends the set $B = \{b_{i,j}\}$ to S.
7. S computes the set $A \bigcap B$ and sends a commitment $com(A \bigcap B)$ to both P_1 and P_2.
8. P_1 sends the set X' and K_2 to S and P_2 sends Y' and K_2 to S.
9. S verified that he received the same key from both parties and that the sets A and B have been computed correctly, namely contain exactly t PRF values for each input element. If the verification fails, S aborts the protocol.
10. S opens the the commitment $com(A \bigcap B)$ and sends the intersection set $A \bigcap B$ to P_1 and P_2.
11. Both P_1 and P_2 verify the open commitment. If the verification fails, they abort the protocol.
12. P_1 checks that the PRP values corresponding to d are present in $A \bigcap B$ and e_1 is not. He also checks that $A \bigcap B$ contains t corresponding PRP values for each element of X' in the intersection $A \bigcap B$. If either of these checks fails, P_1 aborts the protocol.
13. Using K_1 and K_2 P_1 and P_2 recover the values in $X \cap Y$.

Figure 5: Security against any one malicious party.

4. $\mathbf{Sim}_1$ sends to $\mathcal{A}_1$ a commitment of the PRP values from the input set sent by $\mathcal{A}_1$ corresponding to the elements in the set intersection.
5. $\mathbf{Sim}_1$ and $\mathcal{A}_1$ execute the verification where $\mathcal{A}_1$ proves he has submitted exactly t PRP values for each of his inputs. If the verification fails, the simulator aborts the protocol.
6. The simulator opens his commitment and sends the corresponding PRP values to $\mathcal{A}_1$.

The view of $\mathcal{A}_1$ in the simulated and the real executions are identical. In both the real and the simulated execution P_2 receives the correct output if the set submitted by $\mathcal{A}_1$ was formed correctly and otherwise aborts.

□

Claim. The protocol $\big(\mathcal{A}_S[sh], \mathcal{A}_1[nc_S], \mathcal{A}_2[sh]\big)$-securely computes the 2-party set intersection functionality in the $\mathcal{F}_{\mathsf{ct}}$-hybrid model.

The proof of this claim follows from the previous claim and Lemma 6.1.

□

The proofs for the two remaining case when P_2 is malicious have analogous proofs.

■

Multiparty Set Intersection. We observe that both of our protocols can be generalized to setting where multiple parties want to find the intersection of their input sets. In this case the parties agree on common PRF key and then submit the PRF evaluations on their input sets to the server who computes the final intersection. We can apply the same techniques in order to protect against malicious server and malicious parties.

Efficiency. Our set intersection protocol requires that each party performs as many PRF evaluations as the size of his input set. The only works that give two party solutions to the set intersection problem with linear computation complexity in the total size of the input sets are [CKT10] and [JL09], however, the former is in the random oracle model (ROM) and the latter works for input sets of limited size, polynomial in the security parameter and requiring a common random string (CRS). The computation work in both of these solutions includes a linear number of exponentiations equivalent to public key operations. The solution of [HN10] has the best computational complexity while achieving security against malicious adversaries and it requires $O(m + n(\log\log m + p))$ exponentiations where m and n are the sizes of the input sets and p is the bit length of each input element, which is logarithmic in the input domain range. In the multiparty case computation complexity for each party remains the same which improves the computation cost of $O(Nd^2 \log d)$ in the case of N parties with input sets of size d of the most efficient existing solution [DSMRY11].

Acknowledgements

The authors are grateful to Seung Geol Choi for pointing out a mistake in an earlier eprint version of this work. The authors would also like to thank Ben Riva for very helpful discussions, pointers to previous work and suggestions on presentation.

References

[AIK10] B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *International Colloquium on Automata, Languages and Programming (ICALP '10)*, pages 152–163, 2010.

[AKL+09] J. Alwen, J. Katz, Y. Lindell, G. Persiano, a. Shelat, and I. Visconti. Collusion-free multiparty computation in the mediated model. In *Advances in Cryptology - CRYPTO '09*, pages 524–540. Springer-Verlag, 2009.

[ASV08] J. Alwen, a. Shelat, and I. Visconti. Collusion-free protocols in the mediated model. In *Advances in Cryptology - CRYPTO '08*, pages 497–514. Springer-Verlag, 2008.

[BCD+09] P. Bogetoft, D. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach, and T. Toft. Secure multiparty computation goes live. In *Financial Cryptography and Data Security (FC '09)*, pages 325–343. Springer-Verlag, 2009.

[BDJ+06] P. Bogetoft, I. Damgård, T. P. Jakobsen, K. Nielsen, J. Pagter, and T. Toft. A practical implementation of secure auctions based on multiparty integer computation. In *Financial Cryptography and Data Security (FC '06)*, volume 4107 of *Lecture Notes in Computer Science*, pages 142–147. Springer, 2006.

[BDNP08] A. Ben-David, N. Nisan, and B. Pinkas. Fairplaymp: a system for secure multi-party computation. In *ACM Conference on Computer and Communications Security (CCS 2008)*, pages 257–266. ACM, 2008.

[Bea92] D. Beaver. Foundations of secure interactive computing. In *Advances in Cryptology – CRYPTO '91*, pages 377–391. Springer-Verlag, 1992.

[Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *IEEE* 42^{nd} *Annual Symposium on the Foundations of Computer Science (FOCS 2001)*, pages 111–126. IEEE, 2001.

[CCD88] D. Chaum, C. Crépeau, and I. Damgard. Multiparty unconditionally secure protocols. In *ACM symposium on Theory of computing (STOC '88)*, pages 11–19. ACM, 1988.

[CK08] O. Catrina and F. Kerschbaum. Fostering the uptake of secure multiparty computation in e-commerce. In *Conference on Availability, Reliability and Security*, pages 693–700. IEEE Computer Society, 2008.

[CKT10] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *ASIACRYPT*, pages 213–231, 2010.

[CKV10] K.-M. Chung, Y. Kalai, and S. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *Advances in Cryptology - CRYPTO '10*, volume 6223 of *Lecture Notes in Computer Science*, pages 483–501. Springer-Verlag, 2010.

[DI05] I. Damgard and Y. Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *Advances in Cryptology - CRYPTO '05*, volume 3621 of *Lecture Notes in Computer Science*, pages 378–394, 2005.

[DSMRY09] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In *ACNS*, pages 125–142, 2009.

[DSMRY11] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In *ACNS*, pages 130–146, 2011.

[FKN94] Uri Feige, Joe Killian, and Moni Naor. A minimal model for secure computation (extended abstract). In *ACM symposium on Theory of Computing (STOC '94)*, pages 554–563, New York, NY, USA, 1994. ACM.

[FNP04] Michael Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *Proceedings of EUROCRYPT'04*, 2004.

[Gen09] C. Gentry. Fully homomorphic encryption using ideal lattices. In *ACM Symposium on Theory of Computing (STOC '09)*, pages 169–178. ACM Press, 2009.

[GGP10] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: outsourcing computation to untrusted workers. In *Advances in Cryptology - CRYPTO '10*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer-Verlag, 2010.

[GKR08] S. Goldwasser, Y. Kalai, and G. Rothblum. Delegating computation: interactive proofs for muggles. In *Proceedings of the 40th annual ACM symposium on Theory of computing (STOC '08)*, pages 113–122, New York, NY, USA, 2008. ACM.

[GL91] Shafi Goldwasser and Leonid A. Levin. Fair computation of general functions in presence of immoral majority. In *Advances in Cryptology – CRYPTO '90*, pages 77–93. Springer-Verlag, 1991.

[GL02] S. Goldwasser and Y. Lindell. Secure computation without agreement. In *International Conference on Distributed Computing (DISC '02)*, pages 17–32. Springer-Verlag, 2002.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In *ACM Symposium on the Theory of Computation (STOC '87)*, pages 218–229. ACM, 1987.

[Gol04] O. Goldreich. *The Foundations of Cryptography – Volume 2*. Cambridge University Press, 2004.

[HL08] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *TCC*, pages 155–175, 2008.

[HN10] Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. In *Public Key Cryptography PKC 2010*, pages 312–331, 2010.

[IK97] Yuval Ishai and Eyal Kushilevitz. Private simultaneous messages protocols with applications. In *Israel Symposium on the Theory of Computing Systems (ISTCS '97)*, page 174, Washington, DC, USA, 1997. IEEE Computer Society.

[JL09] Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In *TCC*, pages 577–594, 2009.

[KS05] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *CRYPTO*, pages 241–257, 2005.

[LMs05] Matt Lepinksi, Silvio Micali, and abhi shelat. Collusion-free protocols. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, STOC '05, pages 543–552, New York, NY, USA, 2005. ACM.

[LP07] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Proceedings of the 26th annual international conference on Advances in Cryptology (Eurocrypt '07)*, pages 52–78, Berlin, Heidelberg, 2007. Springer-Verlag.

[LPS08] Y. Lindell, B. Pinkas, and N. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In *Proceedings of the 6th international conference on Security and Cryptography for Networks (SCN '08)*, pages 2–20, Berlin, Heidelberg, 2008. Springer-Verlag.

[MF06] P. Mohassel and M. Franklin. Efficiency tradeoffs for malicious two-party computation. In *Conference on Theory and Practice of Public-Key Cryptography (PKC '06)*, volume 3958 of *Lecture Notes in Computer Science*, pages 458–473. Springer, 2006.

[Mic94] S. Micali. Cs proofs. In *IEEE Symposium on Foundations of Computer Science (FOCS '94)*, pages 436–453. IEEE Computer Society, 1994.

[MNPS04] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay—a secure two-party computation system. In *USENIX Security Symposium*, pages 20–20. USENIX Association, 2004.

[MR92] S. Micali and P. Rogaway. Secure computation (abstract). In *Advances in Cryptology - CRYPTO '91*, pages 392–404. Springer-Verlag, 1992.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *ACM Conference on Electronic Commerce (EC '99)*, pages 129–139, New York, NY, USA, 1999. ACM.

[PSSW09] B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure two-party computation is practical. In *Advances in Cryptology - ASIACRYPT '09*, pages 250–267. Springer-Verlag, 2009.

[Yao82] A. Yao. Protocols for secure computations. In *IEEE Symposium on Foundations of Computer Science (FOCS '82)*, pages 160–164. IEEE Computer Society, 1982.

[Yao86] A. Yao. How to generate and exchange secrets. In *IEEE Symposium on Foundations of Computer Science (FOCS '86)*, pages 162–167. IEEE Computer Society, 1986.

A Garbled Circuits

Informally, Garb is considered secure if $(\mathrm{G}(C), \mathrm{G}(x), \mathrm{G}(y))$ reveals no information about x and y. An added property possessed by the construction is *verifiability* which, roughly speaking, means that, given $(\mathrm{G}(C), \mathrm{G}(x), \mathrm{G}(y))$, no adversary can output some $\mathrm{G}(o)$ such that $\mathsf{Translate}(\mathrm{G}(o), \mathrm{T}) \neq f(x, y)$. We discuss these properties more formally in Appendix A.

We recall the properties of Yao's garbled circuit construction which we make use of. These include correctness, privacy and verifiability.

Definition A.1 (Correctness)**.** *We say that* $\mathsf{Garb} = (\mathsf{GarbCircuit}, \mathsf{GarbIn}, \mathsf{Compute}, \mathsf{GarbOut}, \mathsf{Translate})$ *is* correct *if for all functions f, for all circuits C computing f, for all coins $r \in \{0,1\}^\lambda$, and for all x and y in the domain of f*

$$\mathsf{Translate}\Big(\mathsf{Eval}\big(\mathsf{GarbCircuit}(C;r), \mathsf{GarbIn}(C,1,x;r), \mathsf{GarbIn}(C,2,y;r)\big), \mathsf{GarbOut}(r)\Big) = f(x,y).$$

Informally, Garb is considered private if the garbled circuit and the garbled inputs reveal no useful information about x and y.

Definition A.2 (Privacy)**.** *We say that* $\mathsf{Garb} = (\mathsf{GarbCircuit}, \mathsf{GarbIn}, \mathsf{Compute}, \mathsf{GarbOut}, \mathsf{Translate})$ *is* private *if for all functions f, for all circuits C computing f, for all inputs x, y, x' and y' in the domain of f, the following distributions are computationally indistinguishable:*

$$\Big\{\mathsf{GarbCircuit}\big(C;r\big), \mathsf{GarbIn}\big(C,1,x;r\big), \mathsf{GarbIn}\big(C,2,y;r\big)\Big\}$$

and

$$\left\{\mathsf{GarbCircuit}\big(C;r'\big),\mathsf{GarbIn}\big(C,1,x';r'\big),\mathsf{GarbIn}\big(C,2,y';r'\big)\right\},$$

where r and r' are chosen uniformly at random.

Finally, we consider verifiability which, roughly speaking, means that, given a garbled circuit and two garbled inputs, no adversary can find a garbled output that will result in the translation algorithm returning an incorrect output.

Definition A.3 (Verifiability)**.** *We say that* $\mathsf{Garb} = (\mathsf{GarbCircuit}, \mathsf{GarbIn}, \mathsf{Compute}, \mathsf{GarbOut}, \mathsf{Translate})$ *is* verifiable *if for all functions f, for all circuits C computing f, for all inputs x and y in the domain of f, for a uniformly random seed $r \in \{0,1\}^\lambda$, and for all* PPT *adversaries $\mathcal{A}$, the following probability is negligible in k:*

$$\Pr\left[\mathsf{Translate}\big(o',\mathsf{GarbOut}(s)\big) \neq f(x,y) : o' \leftarrow \mathcal{A}\big(\mathsf{GarbCircuit}(C;r),\mathsf{GarbIn}(C,1,x;r),\mathsf{GarbIn}(C,2,y;r)\big)\right]$$

where the probability is over the coins of $\mathcal{A}$.

B Secure Delegated Computation

A delegated computation scheme consists of four polynomial-time algorithms $\mathsf{Del} = (\mathsf{Gen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ that work as follows. Gen is a probabilistic algorithm that takes as input a security parameter k and a function f and outputs a public and secret key pair $(\mathrm{PK}, \mathrm{SK})$ such that the public key encodes the target function f. $\mathsf{ProbGen}$ is a probabilistic algorithm that takes as input a secret key SK and an input x in the domain of f and outputs a public encoding σ_x and a secret state τ_x. $\mathsf{Compute}$ is a deterministic algorithm that takes as input a public key PK and a public encoding σ_x and outputs a public encoding σ_y. Verify is a deterministic algorithm that takes as input a secret key SK, a secret state τ_x and a public encoding σ_y and outputs either an element y of f's range or the failure symbol $\perp$.

We recall the formal definitions of correctness, verifiability and privacy for a delegated computation scheme.

Definition B.1 (Correctness)**.** *A delegated computation scheme* $\mathsf{Del} = (\mathsf{Gen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ *is* correct *if for all functions f, for all* PK *and* SK *output output by* $\mathsf{Gen}(1^k, f)$*, for all x in the domain of f, for all σ_x and τ_x output by* $\mathsf{ProbGen}_{\mathrm{SK}}(x)$*, for all σ_y output by* $\mathsf{Compute}_{\mathrm{PK}}(\sigma_x)$*,* $\mathsf{Verify}_{\mathrm{SK}}(\tau_x, \sigma_y) = f(x)$*.*

A delegated computation scheme is *verifiable* if a malicious worker cannot convince the client to accept an incorrect output. In other words, for a given function f and input x, a malicious worker should not be able to find some σ' such that the verification algorithm outputs $y' \neq f(x)$. This intuition is formalized in the following definition.

Definition B.2 (Verifiability)**.** *Let* $\mathsf{Del} = (\mathsf{Gen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ *be a delegated computation scheme, $\mathcal{A}$ be an adversary and consider the following probabilistic experiment* $\mathsf{Ver}_{\mathsf{Del},\mathcal{A}}(k)$*:*

1. *the challenger computes* $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^k, f)$*,*
2. *let $\mathcal{O}(\mathrm{SK}, \cdot)$ be a probabilistic oracle that takes as input an element x in the domain of f, computes* $(\sigma, \tau) \leftarrow \mathsf{ProbGen}_{\mathrm{SK}}(x)$ *and outputs σ,*
3. *given* PK *and oracle access to $\mathcal{O}(\mathrm{SK}, \cdot)$, $\mathcal{A}$ outputs an input x,*

4. *the challenger computes* $(\sigma_x, \tau_x) \leftarrow \mathsf{ProbGen}_{\mathrm{SK}}(x)$,

5. *given* σ_x, *the adversary* $\mathcal{A}$ *outputs an encoding* σ',

6. *if* $\mathsf{Verify}_{\mathrm{SK}}(\tau, \sigma') \notin \{\perp, f(x)\}$ *then output* 1 *else output* 0.

We say that Del *is verifiable if for all* PPT *adversaries* $\mathcal{A}$,

$$\Pr\left[\,\mathsf{Ver}_{\mathsf{Del},\mathcal{A}}(k) = 1\,\right] \leq \mathsf{negl}(k)$$

where the probability is over the coins of Gen, $\mathcal{O}$, $\mathcal{A}$ *and* ProbGen.

Informally, a delegated computation scheme is private if its public encodings reveal no useful information about the input x.

Definition B.3 (Privacy)**.** *Let* $\mathsf{Del} = (\mathsf{Gen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ *be a delegated computation scheme,* $\mathcal{A}$ *be a stateful adversary and consider the following probabilistic experiment* $\mathsf{Priv}_{\mathsf{Del},\mathcal{A}}(k)$*:*

1. *the challenger computes* $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^k, f)$,

2. *let* $\mathcal{O}(\mathrm{SK}, \cdot)$ *be a probabilistic oracle that takes as input an element* x *in the domain of* f*, computes* $(\sigma, \tau) \leftarrow \mathsf{ProbGen}_{\mathrm{SK}}(x)$ *and outputs* σ,

3. *given* PK *and oracle access to* $\mathcal{O}(\mathrm{SK}, \cdot)$, $\mathcal{A}$ *outputs two inputs* x_0 *and* x_1,

4. *the challenger samples a bit* b *at random and computes* $(\sigma_b, \tau_b) \leftarrow \mathsf{ProbGen}_{\mathrm{SK}}(x_b)$,

5. *given* σ_b, *the adversary* $\mathcal{A}$ *outputs a bit* b',

6. *if* $b' = b$ *output* 1 *else output* 0.

We say that Del *is private if for all* PPT *adversaries* $\mathcal{A}$,

$$\Pr\left[\,\mathsf{Priv}_{\mathsf{Del},\mathcal{A}}(k) = 1\,\right] \leq \mathsf{negl}(k)$$

where the probability is over the coins of Gen, $\mathcal{O}$, $\mathcal{A}$ *and* ProbGen.