Secure Data Exchange: A Marketplace in the Cloud

Ran Gilad-Bachrach[1], Kim Laine[1], Kristin Lauter[1], Peter Rindal[1,2], and Mike Rosulek[1,2]

[1]Microsoft Research, Redmond, USA
[2]Oregon State University, Oregon, USA

June 14, 2016

Abstract

A vast amount of data belonging to companies and individuals is currently stored *in the cloud* in encrypted form by trustworthy service providers such as Microsoft, Amazon, and Google. Unfortunately, the only way for the cloud to use the data in computations is to first decrypt it, then compute on it, and finally re-encrypt it, resulting in a problematic trade-off between value/utility and security. At a high level, our goal in this paper is to present a general and practical cryptographic solution to this dilemma.

More precisely, we describe a scenario that we call *Secure Data Exchange* (SDE), where several data owners are storing private encrypted data in a semi-honest non-colluding cloud, and an evaluator (a third party) wishes to engage in a secure function evaluation on the data belonging to some subset of the data owners. We require that none of the parties involved learns anything beyond what they already know and what is revealed by the function, even when the parties (except the cloud) are active malicious. We also recognize the ubiquity of scenarios where the lack of an efficient SDE protocol prevents for example business transactions, research collaborations, or mutually beneficial computations on aggregated private data from taking place, and discuss several such scenarios in detail.

Our main result is an efficient and practical protocol for enabling SDE using *Secure Multi-Party Computation* (MPC) in a novel adaptation of the server-aided setting. We also present the details of an implementation along with performance numbers.

1 Introduction

1.1 Motivation

Cloud storage is becoming the *de facto* way for businesses to manage their growing stockpiles of data, with an incredible amount of data already being stored in the cloud: Microsoft's Azure service alone has over 50 trillion objects.[1] Basic security standards require data to be encrypted both *in transit* to or from the cloud, and when it remains *at rest* in the cloud. Yet data at rest has only limited value. Being able to *compute* on the encrypted data without having to decrypt it first would massively increase its utility, and in some cases enable entirely new markets for cloud technologies. Unfortunately computing on encrypted data is notoriously difficult, often requiring highly sophisticated and costly cryptographic techniques such as *homomorphic encryption.* Currently the standard approach is to perform the computations on unencrypted data, resulting in an apparent trade-off between utility and privacy. Furthermore, users of cloud storage list security

[1] http://www.businessinsider.com/microsoft-azure-usage-doubled-2015-4

of their data as their biggest concern[2], and that concern is significantly amplified when the data is used for computations. Hence, at a high level, we will address the following question in this paper:

> *What is the best way to perform useful computations on the huge amounts of data already stored in the cloud, while preserving privacy to the greatest possible extent?*

A practical and adoptable solution should satisfy at least the following requirements:

- The system should **leverage the existing cloud storage infrastructure.** Cloud service providers are already equipped to store the data of their customers, so data should either remain stored in its existing form, or at least in some other "reasonable" form that causes little or no extra overhead in the cloud storage costs. An example of an "unreasonable" requirement would be an encoding/encryption that is 128 times larger than the plaintext data. We would like to point out that this requirement immediately rules out most forms of homomorphic encryption.

  Whether encrypted or unencrypted, data in the cloud must be *persistent* in the sense that it can be stored for an arbitrarily long period of time, and *updatable* so that the data owners can easily append to it, or ask the cloud to delete parts of it.

- The system should **align to the existing incentives for cloud services.** Users store their data in the cloud to avoid managing their own storage solutions on site and to benefit from collective economies of scale.

  In a system for computations in the cloud, often there is one party with the majority of interest in the outcome of the computation. That party, along with the cloud provider, are the only ones willing to expend significant effort to carry out computations with a cryptographic security guarantee. Other parties, whose data might be involved in the computation, should have only minimal involvement in the computation (e.g., only to authorize a computation). As a corollary, data should not require expensive maintenance in order to maintain security – the same data should be reusable for many computations with different parties.

- The system should **use trust models that reflect the current reality of cloud services.** Users of cloud storage place some trust in the cloud service providers, but that trust is limited. Sensitive data can be encrypted before being stored in the cloud, reflecting a threat model in which the cloud provider is considered *semi-honest*[3]. Tools such as *proofs of retrievability* [13], which protect against more severe *active malicious* behavior by the cloud provider, are rare in practice. While protection against fully malicious cloud providers would be ideal, the reality is that users with this perception of cloud providers are unlikely to be using the cloud for storage anyway.

  The system should leverage this limited trust in the cloud provider to reduce the cost of computations as much as possible. Of course, some security requirements are non-negotiable: the data owners should have absolute control over how their data is being used.

Some readers may immediately recognize *Secure Multi-Party Computation* (MPC) (see e.g. [31, 20]) as the "textbook" solution to the scenario described above. Indeed, our solution is based

[2] http://searchcloudstorage.techtarget.com/feature/More-companies-turn-to-cloud-storage-service-providers

[3] *Semi-honest* adversaries follow the protocol but attempt to learn more than their intended share of information from their view of the protocol execution. *Malicious* adversaries can deviate arbitrarily from the protocol. The cloud is *non-colluding* if the messages it sends to the other parties reveal no information about the cloud's input other than what can be learned from the output of the function.

on MPC techniques, yet we believe that existing MPC research does not address many of the central aspects of our setting. For example, MPC does not naturally provide reusable encryption of the data of any of the parties involved, and as such is non-trivial to integrate with secure cloud storage. In addition, in plain MPC all of the parties involved typically have to participate in an online phase with a linear amount of work and communication, which is in practice often unacceptable.

In this paper we will describe a practical protocol that allows an arbitrary number of *data owners* to store data in encrypted form to a cloud service in a persistent and updatable manner, and allows a third party (an *evaluator*) to compute a function on the data. The result of the function can be shared with any subset of the parties involved, and none of the parties will learn anything about the data beyond what they already know and what will be revealed by the function output. The cloud learns nothing. The data stored in the cloud can be used repeatedly for an arbitrary number of such interactions. In section 4 we prove that our solution remains secure in the presence of malicious data owners and/or evaluator, as long as the cloud remains semi-honest and does not collude with the evaluator. We call this scenario *Secure Data Exchange* (SDE).

1.2 Secure Data Exchange

To further motivate our construction, consider the following realistic business scenarios that face severe and perhaps insurmountable difficulties due to data privacy issues:

- A pharmaceutical company would like to purchase anonymized patient medical records from several hospitals for research purposes. Since the price of such medical data is typically very high, the pharmaceutical company would like to have a certain confidence in the quality and usefulness of the data before agreeing to buy it. The sellers are not willing to share the data with the buyer before a deal has been agreed upon. Even if the data would not be maximally interesting, the buyer might agree to buy it at a lower cost, but such a negotiation is again difficult without the seller sharing precise information about the data. One solution used in practice is for the seller to agree to compute certain statistics on the data, but this typically provides too low of a resolution for the buyer to make a truly informed decision.

- A medical center would like to compare the expected outcome of its treatment plan for pneumonia with the expected outcomes of the treatment plans used at competing medical centers. The problem is that no-one wants to publicly disclose such information for the fear of being *called out* for providing less effective care.

- A company is developing machine learning models that try to assist primary care providers in choosing the best treatment plans for their patients in a variety of situations. The company would like to buy anonymized patient medical records from hospitals to further develop and study their models, but only if their data does *not* already fit the model well enough. This could in theory be tested by running simple statistical tests comparing the model parameters with the data, but in practice not because the hospital is not willing to disclose its data before a deal has been made.

- A big enterprise wants to outsource a part of its internal services to an external provider. The legal and practical costs for realizing such a partnership might be so enormous, e.g. for making sure that strategic data of the enterprise will not get disclosed by the external provider, that it would end up being more efficient for the enterprise to acquire its own resources for providing the service *in house*.

- A company producing chocolate bars would like to learn detailed information about the chocolate bar market (e.g. market elasticity) by combining its own data with the data of other companies in the same or related market. Its goal would be to reduce costs through improved efficiency and better pricing, but unfortunately the other companies are not willing to share their private financial data.

- In the near future when genomic sequencing is projected to become even more commonplace than it is today, it is conceivable that individual people would like to have an opportunity to sell *access* to a part of their genomic data to trustworthy companies (e.g. pharmaceutical companies) or research groups, who are willing to pay an appropriate price. Such individuals would like to upload their data in encrypted form to a special marketplace, which potential buyers could then run aggregated and anonymized query at will. This is somewhat similar to what *Project Beacon*[4] provides, with the significant distinction that Project Beacon operates on a distributed network of hospitals and as such is not easily accessible to individuals.

For the examples above, current solutions that are actually used in practice require substantial and costly litigation to preserve the interests of each party, while still typically failing to preserve full privacy. In some scenarios anonymization procedures end up causing the resolution of the data to decrease so much that a significant part of its value is lost in the process. Instead, we observe that each of these scenarios can be framed in terms of SDE, which in turn is enabled by our general solution.

In some instances (see the examples above) the SDE framework can be viewed as a particular type of a *reverse auction* with extra security and privacy measures, or a *secure marketplace* where several *sellers* (data owners in Section 1.1) have valuable data they wish to sell, and have uploaded it in the cloud in encrypted form to put it *on the market*, and a *buyer* (evaluator in Section 1.1) wants to buy data from one or more of the sellers but only if it satisfies certain conditions. In typical situations the price the buyer would offer depends on some particular qualities of the data, and sellers might want to only agree if the price offered is above some threshold, so a negotiation on the value of private data must take place. In some cases the buyer would prefer to keep the price it is willing to offer secret, and the sellers would not want to reveal their conditions for accepting or rejecting offers. In situations with more than one seller, the buyer might want to only engage in a deal with a particular seller, or sellers, whose data they determine to be of most use to the buyer, whose price is the lowest, whose data has been on the market for the shortest/longest time, or any combination of properties of this type. In a slightly more general situation, the buyer who might not be interested in buying the data itself, but only some limited number of bits of information about it, e.g. the value of a particular function evaluated on it. In this case the price might depend both on the function and on the bit width of the output.

As was briefly mentioned in Section 1.1, our solution for enabling SDE is based on Secure Multi-Party Computation (MPC), which in its most basic form allows two or more parties to evaluate a function on their private inputs in such a way that one or more of the parties obtains the output of the function, but none of the parties learns anything about each other's inputs, except what can perhaps be inferred from the output of the function. While classic MPC can easily meet some of the requirements of SDE (recall Section 1.1), it falls far from meeting all of them.

To overcome the limitations of classic MPC, and motivated by the scenarios described above, we are naturally led to consider a different setting where a semi-honest and non-colluding cloud assists in the MPC and does not contribute any input of its own, nor receive any output. Such a *server-aided* setting has been extensively studied and shown to be efficient (see [15, 14, 10]). Our starting

[4] https://beacon-network.org

point is similar to that of e.g. [15], but we develop the server-aided approach in a new direction that we find to be strongly motivated by practical applications. In particular, the security model that we focus on maintains data privacy even if everyone except the cloud is arbitrarily malicious, but is still flexible enough to allow for the powerful features described in Section 1.1. We implement several applications and evaluate their performance to demonstrate the practicality of our solution in section 5.

We are not aware of any earlier work that would focus on SDE as a fundamental tool for solving a wide variety of data privacy issues in business transactions. In fact, we believe that an efficient implementation of the SDE framework, such as the one that we present, can significantly enhance existing and open up entirely new business opportunities in areas that have earlier been unrewarding or impossible due to privacy concerns.

1.3 Overview of our Protocol

We start our overview with a simplified SDE protocol that achieves security against semi-honest adversaries (with a non-collusion assumption). The parties involved are denoted as follows: $\mathcal{C}$ (cloud), $\mathcal{P}_1, \ldots, \mathcal{P}_n$ (data owners), and $\mathcal{Q}$ (a third party/function evaluator). The input data of a party $\mathcal{P}_i$ is denoted by $\mathbf{x}_i$ and any input data of $\mathcal{Q}$ by $\mathbf{x}_\mathcal{Q}$.

In many of the examples of Section 1.2 the data $\mathbf{x}_i$ is meaningful to use for several executions of the protocol with different $\mathcal{Q}$, however, it is also possible for the $\mathcal{P}_i$ to have *per computation* inputs analogous to $\mathbf{x}_\mathcal{Q}$, but we omit this simple addition in our discussion here.

The semi-honest protocol works as follows. First, each $\mathcal{P}_i$ uniformly samples a secret seed $r_i \leftarrow \{0,1\}^\kappa$ and computes $\mathbf{z}_i := \mathbf{x}_i \oplus g(r_i)$, where g is a *pseudo-random function* (PRF) that all parties have agreed to use. Then each $\mathcal{P}_i$ uploads its secret-share $\mathbf{z}_i$ to $\mathcal{C}$ for long-term storage.

When a party $\mathcal{Q}$ wishes to engage in SDE with some subset of the parties $\mathcal{P}_i$, it will ask those particular $\mathcal{P}_i$ for their respective seeds r_i. After all involved parties have agreed on a function $f(\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{x}_\mathcal{Q})$ to be computed, $\mathcal{C}$ and $\mathcal{Q}$ engage in a secure 2PC protocol where the private input of $\mathcal{C}$ is the set of the $\mathbf{z}_i$ and the private input of $\mathcal{Q}$ is the set of the $g(r_i)$ together with its own private data $\mathbf{x}_\mathcal{Q}$. The secret-shares $\mathbf{z}_i$ and $g(r_i)$ are combined only *inside* the 2PC protocol.

We point out several important aspects of this approach:

- While we use the language of secret sharing to describe $\mathbf{z}_i = \mathbf{x}_i \oplus g(r_i)$, in practice $g(r_i)$ will be AES in counter-mode keyed by r_i. Hence $\mathbf{z}_i$, which is what $\mathcal{P}_i$ uploads to the cloud, is in fact a standard AES-CTR encryption of $\mathbf{x}_i$ under an ephemeral key.

- By choosing g to be AES in counter mode, it becomes easy to perform computations on any *subset* of the data $\mathbf{x}_i$. The cloud $\mathcal{C}$ provides the appropriate subset of $\mathbf{z}_i$ while the evaluator $\mathcal{Q}$ can compute the subset of $g(r)$ with random access. We note that this approach leaks to all parties (including the server) which subset of the data is used. However, it is well-known that hiding this access pattern would require all parties to "touch" every bit of their shares.

- Our goal is to reduce the burden on each $\mathcal{P}_i$ as much as possible, beyond simply uploading their data to the cloud. By letting one of the shares $g(r_i)$ be pseudorandom, the communication from $\mathcal{P}_i$ and $\mathcal{Q}$ is reduced to a constant (independent of $|\mathbf{x}_i|$).

- By exploiting the linearity of the secret-share reconstruction, it is possible fold it into the OT extension such that the labels encoding $g(r_i)$ need not be sent.

We now describe our enhancements to this basic protocol, making it secure against malicious $\mathcal{Q}$ and $\mathcal{P}_i$, and a semi-honest cloud $\mathcal{C}$. The change results in a security/performance trade-off which preserves the practicality of the protocol.

Preventing $\mathcal{Q}$ from cheating in the 2PC with $\mathcal{C}$: When $\mathcal{Q}$ is potentially malicious, it may try to cheat in the 2PC protocol execution with $\mathcal{C}$. The natural way to address this is to use the *garbled circuits* paradigm of Yao [31, 20]. This protocol paradigm is naturally secure against a malicious receiver.

Forcing $\mathcal{Q}$ to use the correct inputs: In Yao's protocol between $\mathcal{C}$ and $\mathcal{Q}$, $\mathcal{C}$ produces a so-called *garbled circuit.* A garbled circuit can be thought of as an encryption of a normal Boolean circuit, where all wires hold encrypted (garbled) binary values. The garbled output of the garbled circuit can be decrypted using keys known only to the garbler, i.e. the cloud $\mathcal{C}$. To evaluate the garbled circuit, $\mathcal{Q}$ must obtain garbled inputs corresponding to its own input to the computation. This is typically done using *oblivious transfer* (OT) (see Section 2.1 and Section 2.2 for details).

$\mathcal{Q}$ is supposed to use $g(r_i)$ as its input to the 2PC, and we need a way for the data owners $\mathcal{P}_i$ to bind $\mathcal{Q}$ to these inputs in the oblivious transfers. We achieve this by a novel adaptation of the malicious-secure *OT extension* (OT-e) protocol of [16] (see also [12, 26, 1, 2]). At a very high level, $\mathcal{P}_i$ can execute the OT-e protocol with $\mathcal{C}$ in an offline phase, with $\mathcal{P}_i$ acting as the oblivious receiver (OT receiver). There is a point in the protocol at which $\mathcal{P}_i$ would send a large message that binds it to $g(r_i)$ as its choice bits. Rather than send this message, we have $\mathcal{P}_i$ commit to it.

At this point, the receiver's internal state in the OT-e protocol can be derived from a short seed (and its input $g(r_i)$). For an SDE computation, $\mathcal{P}_i$ sends this short seed to $\mathcal{Q}$, who can then reconstruct the state and continue the OT-e protocol. The commitment made by $\mathcal{P}_i$ ensures that $\mathcal{Q}$ must complete the OT-e protocol using $g(r_i)$ as its OT selection string.

This approach to ensuring correct inputs has the following advantages:

- $\mathcal{P}_i$ must *compute* the long message for the OT-e protocol (it has length $O(\kappa|\mathbf{x}_i|)$), but $\mathcal{P}_i$ does not need to *send* it. Only a short commitment (κ bits, in the random oracle model) needs to be sent to $\mathcal{C}$, and a short seed (κ bits) sent to $\mathcal{Q}$.

- $\mathcal{P}_i$'s involvement in the OT-e input-binding technique happens only once, i.e. at the time it uploads the data to the cloud. We show that performing a single OT-e instance suffices for using the same dataset in many SDE computations.

- By exploiting the internal structure of the OT-e protocols, we can allow the SDE computations to use only certain subsets of the data. In short, $\mathcal{P}_i$ makes a commitment to a long OT-e message, whose blocks have a 1-to-1 correspondence to the bits of $g(r_i)$. Hence, if the commitment takes the form of a Merkle tree, then $\mathcal{Q}$ can efficiently and selectively decommit to only a specified region of this OT-e message.

Overall, $\mathcal{P}_i$'s involvement in the SDE protocol is minimal. While uploading data to the cloud, $\mathcal{P}_i$ performs some additional cryptographic processing. However, the data uploaded to the cloud is simply an AES-CTR encryption of its data. When $\mathcal{Q}$ wishes to perform a computation on $\mathcal{P}_i$'s data, $\mathcal{P}_i$ indicates its approval to $\mathcal{C}$ somehow, and then just sends a very short $O(\kappa)$-bit message to $\mathcal{Q}$.

1.3.1 Security

We describe the security of our final protocol in more detail in Section 4. For now we merely mention that the protocol is secure under the assumptions that:

- The cloud $\mathcal{C}$ is semi-honest, and

- Any number of $\mathcal{P}_1, \ldots, \mathcal{P}_n, \mathcal{Q}$ can be potentially malicious, and

- The cloud $\mathcal{C}$ and evaluator $\mathcal{Q}$ do not collude.

To formally model a setting in which adversaries do not collude, we use the definitions and results of [14].

1.4 Related Work

Our work is most closely related to protocols in the server-aided MPC model, epitomized by the *Salus* protocol of Kamara et al. [15]. Salus, and other protocols in this model [8, 7, 23, 9], consider weak parties who wish to outsource as much of an MPC computation to a powerful cloud server as possible. The server itself has typically no input of its own, and it receives no output from the MPC. In this sense the goals of Salus are partially similar to what we are trying to achieve. In fact, [15] describes a protocol similar to the semi-honest variant of our protocol (see Section 1.3 below), but with an obviously different focus.

However, our work deviates significantly from this line of work in how the protocol is adapted to a particular use case. The main differences are:

(1) Difference in security model: In the server-aided protocols listed above, the cloud is not assumed to be semi-honest. This difference results in a severe asymptotic and concrete efficiency penalty, and makes a direct performance comparison less valuable. Our focus on this weakened security model stems not only from considerable performance gains, but also from the real-world applications that this work addresses. Foremost in our considerations is the need for a highly flexible framework and the ability to leverage cloud storage in a secure manner.

The security of the Salus protocol is proved under two settings: one in which the server is *covert* [3] and non-colluding, and another one in which the server is malicious and non-colluding, and all but the circuit evaluator are malicious. To achieve security against the server, Salus uses *cut-and-choose* [25, 21, 22, 17, 30, 19] which results in as much as a 40× overhead in communication and computation. Despite this relaxation of the security model, Salus requires a strong non-collusion assumption. Even in the presence of a semi-honest adversary, if any party colludes with the server, all of the parties' inputs are trivially leaked. Our protocol relaxes this non-collusion assumption to only apply between the cloud and the evaluator.

The Whitewash protocol [7] considers the malicious setting with only two parties where one outsources their work to a cloud. Whitewash achieves superior security than Salus in that the party who outsources their work can not learn the other parties input when colluding with the cloud. However, none of the protocols protect against a cloud colluding with the circuit evaluator due to the server aided paradigm reducing to two party secure computation.

(2) The protocols in this model (with one exception — see below) do not consider long-term storage and repeated use of the data of the weak parties, and as such is not directly applicable to the SDE setup. In particular, in the Salus protocol all parties must perform a bandwidth-intensive *Distributed OT* protocol for each computation. Furthermore, Salus does not specify any mechanism for allowing parties to receive private outputs but instead assumes that each party receives the same output. Trivially, any secure computation protocol can provide private output by encrypting (e.g. one time pad) each party's output under an input key. However, this results in additional overhead that breaks our requirement for data owners to perform a constant amount of online work.

The work of Mood et al. [23] does consider long-term storage of MPC data. However, the data is represented between computations in *garbled form* (i.e., as wire labels) which incurs a *multiplicative* storage overhead of κ (e.g., 128) bits. For very large data, of the type that may be outsourced to the cloud, such an overhead would be prohibitive.

2 Preliminaries

2.1 Garbled Circuits

Conceived in the seminal work of Yao [31], *garbled circuits* allow two parties with respective private inputs $\mathbf{x}$ and $\mathbf{y}$ to jointly compute a possibly probabilistic functionality $f(\mathbf{x}, \mathbf{y}) = (f_1(\mathbf{x}, \mathbf{y}), f_2(\mathbf{x}, \mathbf{y}))$, such that the first party learns $f_1(\mathbf{x}, \mathbf{y})$ and the second party learns $f_2(\mathbf{x}, \mathbf{y})$. Garbled circuits have become fundamental building blocks in many cryptographic protocols in recent years, and while they are most widely used in two-party secure function evaluation, other multi-party protocols make extensive use of them (see e.g. [27, 11]).

Formalizing appropriate security definitions and proving the security of Yao's protocol under these definitions was a difficult task, and was finally completed in 2009 by Lindell and Pinkas [20]. Intuitively, the security requirement is that no information is learned by either party beyond their prescribed output (*privacy*) and that the output distribution follows that specified by f (*correctness*).

The garbled circuits construction can be thought of as a compiler which takes a functionality f as input and outputs a secure protocol for computing f. First, the functionality is expressed as a Boolean circuit C consisting of gates (typically AND and XOR gates). Each gate g takes two logical bits $a, b \in \{0, 1\}$ as inputs and returns a logical bit $c := g(a, b)$ as output. The secure protocol then evaluates each gate of the circuit C such that it hides the logical values in all internal *wires* and contains a mechanism to decode the garbled output wires.

The first party known as the *garbler* generates the garbled wires and the garbled gates. The other party, known as the *evaluator*, needs to obtain the garbled wire labels from the garbler for their respective input. To ensure the privacy of the evaluator's input, this must be done without revealing to the garbler which labels the evaluator picks. In addition, the evaluator must be prevented from evaluating the garbled circuit on several inputs, so for each garbled wire the evaluator is allowed to learn precisely one of the two labels. This is achieved using *Oblivious Transfer* (OT), which we will discuss in 2.2. Once the evaluator has learns the input wire labels for a garbled gate, exactly one garbled output wire labels can be learned. A garbled circuit is the collection of all the garbled gates and may be evaluated with an input encoding, i.e. one label per wire. The above process can then be repeatedly applied to each gate of the garbled circuit.

For security of the garbled circuits construction, the evaluator will need to learn exactly one of the two output wire labels C_0, C_1, while the other one must remain entirely unknown. It can then be seen that the use of *malicious secure* OT (see 2.2) yields a protocol that is secure against a malicious evaluator who may arbitrarily deviate from the protocol. However, the garbler can maliciously construct a garbled gate or an entire circuit that computes the wrong logic. The evaluator may not be able to detect such malicious behavior, and it can be shown that all security properties of the construction are lost. A standard technique for overcoming this is known as *cut-and-choose*, in which the garbler generates several garbled circuits and sends them to the evaluator. The evaluator randomly checks some of them for correctness, and if all turn out to be honestly generated the evaluator evaluates the remaining ones. When the evaluator checks and evaluates $O(\kappa)$ of the circuits, the garbler only succeeds in this attack with exponentially small probability in the security parameter κ.

Due to the significant overhead incurred in sending several garbled circuits, in this work we will avoid the use of cut-and-choose and instead assume that the garbler is semi-honest and garbles the correct circuit. In particular, the cloud $\mathcal{C}$ will take the role of garbler and receive no output.

There are several ways of improving the practicality and efficiency of garbled circuits. Most importantly, XOR gates can be evaluated *for free* using the Free-XOR technique of [18], and garbled

AND gates can be reduced down to two 128 bit AES blocks using the *half-gates* construction of [32]. Finally, note that it is not necessary to know the full garbled output labels for decoding, as long as they are known to differ at a particular (typically the last) bit position. This makes it very easy for the garbler to communicate the decoding information to the evaluator, namely all the garbler needs to do is to give the evaluator a string of permutation bits that tells the evaluator how the last bit of a particular garbled output label corresponds to the logical output value [21].

2.2 OT Extension

Oblivious Transfer (OT) is a fundamental cryptographic primitive with numerous applications to modern protocols. As was explained in 2.1, we will use OT for communicating wire labels in Yao's protocol from the garbler to the evaluator, without revealing the evaluator's input to the garbler. The idea is that a *sender* $\mathcal{S}$ has two input strings $\mathbf{x}_0$ and $\mathbf{x}_1$ of length ℓ, and a *receiver* $\mathcal{R}$ has a *selection bit* $b \in \{0,1\}$, and wants to obtain $\mathbf{x}_b$ from $\mathcal{S}$ in an *oblivious way*, meaning that $\mathcal{S}$ does not learn b and $\mathcal{R}$ is guaranteed to obtain only $\mathbf{x}_b$ and learns no information about $\mathbf{x}_{1-b}$. Figure 1 describes an ideal functionality for the oblivious transfer primitive.

Parameters: A sender $\mathcal{S}$ and a receiver $\mathcal{R}$.

Main Phase: On input $(\textsc{select}, \mathsf{sid}, b)$ from $\mathcal{R}$ and $(\textsc{send}, \mathsf{sid}, (\mathbf{x}_0, \mathbf{x}_1))$ from $\mathcal{S}$, send $\mathcal{R}$ $(\textsc{recv}, \mathsf{sid}, \mathbf{x}_b)$.

Figure 1: $\mathcal{F}_{\mathsf{ot}}$ ideal functionality.

While one round of OT is fairly efficient to do [28], it requires public-key primitives and as such is not practical for exchanging very large amounts of information. For example, in Yao's protocol if the bit-length of the evaluator's input is ℓ and each wire label has length κ (typically the labels are AES blocks and $\kappa = 128$), the evaluator must engage in ℓ OTs with the garbler. This is problematic when ℓ is large, so a technique called *OT extension* was invented (see [12, 26, 1, 2, 16]) to efficiently extend κ so-called *base OT*s into ℓ OTs. More precisely, instead of having to perform ℓ OTs of length κ, it will be enough to perform κ OTs of length κ. In this section we will explain how OT extension works following [16]. Subsequently we will describe a novel adaptation of OT extension to force the evaluator in Yao's protocol to either remain honest, or with overwhelming probability to fail the garbled circuit evaluation. In this description we will use OT as a black box and focus on explaining the extension procedure.

We break the OT extension protocol into several steps. The protocol we describe below is a slight variant of what is referred to as *correlated OT extension* in [16]. It is not secure against an active malicious $\mathcal{R}$, but is easily strengthened to be by adding a minimal amount of overhead, as is also described in [16]. For the sake of simplicity we will omit describing the malicious secure OT extension protocol.

Let $\{(\mathbf{x}_0^i, \mathbf{x}_1^i)\}$ for $i = 1, \ldots, \ell$ be pairs of κ-bit messages that $\mathcal{S}$ wants to obliviously transfer to $\mathcal{R}$. In other words, $\mathcal{R}$ has an ℓ-bit selection string $\mathbf{r} := (r_1, \ldots, r_\ell)$ and it wants to obtain the messages $\mathbf{x}_{r_i}^i$ in an oblivious way. The semi-honest OT extension protocol $\Pi_{\mathsf{OT\text{-}e}}$ that we will use is described in Figure 2.

It will be important to understand the amount of communication between $\mathcal{R}$ and $\mathcal{S}$ in each step. In the Setup Phase a small amount of OT communication between $\mathcal{R}$ and $\mathcal{S}$ takes place. Note that typically we would take $\kappa := 128$. In the Select and Receive Phases a significant amount of communication takes place. Matrices of size $\ell \times \kappa$ are sent between $\mathcal{R}$ and $\mathcal{S}$, where ℓ can potentially be very large.

Parameters: A sender $\mathcal{S}$ and receiver $\mathcal{R}$. A security parameter κ, and a number ℓ (the number of OTs to be obtained).

Setup Phase: On common $(\textsc{setup}, \mathsf{sid}, \kappa)$: $\mathcal{R}$ uniformly samples $\mathbf{k}_b^i \leftarrow \{0,1\}^\kappa$ for each $i \in [\kappa], b \in \{0,1\}$, and $\mathcal{S}$ uniformly samples $\Delta \leftarrow \{0,1\}^\kappa$.

For each $i \in [\kappa]$, $\mathcal{R}$ sends $(\textsc{send}, (\mathsf{sid}, i), (\mathbf{k}_0^i, \mathbf{k}_1^i))$, and $\mathcal{S}$ sends $(\textsc{select}, (\mathsf{sid}, i), \Delta_i)$ to $\mathcal{F}_{\mathsf{ot}}$, who responds with $(\textsc{recv}, (\mathsf{sid}, i), \mathbf{k}_{\Delta_i}^i)$.

Select Phase: Let $\mathbf{t}_b^i := g(\mathbf{k}_b^i)$. On input $(\textsc{select}, \mathsf{sid}, \mathbf{r})$ from $\mathcal{R}$, where $\mathbf{r}$ is an ℓ-bit selection string, $\mathcal{R}$ computes a matrix $\mathbf{u} \in \{0,1\}^{\ell \times \kappa}$ whose i-th column is $\mathbf{u}^i := \mathbf{t}_0^i \oplus \mathbf{t}_1^i \oplus \mathbf{r}$, and sends it to $\mathcal{S}$. For each $i \in [\kappa]$, $\mathcal{S}$ computes ℓ-bit column vectors $\mathbf{q}^i := (\Delta_i \cdot \mathbf{u}^i) \oplus \mathbf{t}_{\Delta_i}^i = \mathbf{t}_0^i \oplus (\Delta_i \cdot \mathbf{r})$.

Receive Phase: On input $(\textsc{send}, \mathsf{sid}, (i, \mathbf{x}_0^i, \mathbf{x}_1^i))$ from $\mathcal{S}$, let $\mathbf{t}_i$ (resp. $\mathbf{q}_i$) denote the i-th row of the $\ell \times \kappa$ matrix formed from the column vectors $\mathbf{t}_0^j$ (resp. $\mathbf{q}^j$) for each $j \in [\kappa]$. $\mathcal{S}$ sends

$$(i, \mathbf{y}_0^i, \mathbf{y}_1^i) := \big(i, \mathbf{x}_0^i \oplus H(i, \mathbf{q}_i), \mathbf{x}_1^i \oplus H(i, \mathbf{q}_i \oplus \Delta)\big)$$

to $\mathcal{R}$, who computes $\mathbf{x}_{r_i}^i = \mathbf{y}_{r_i}^i \oplus H(i, \mathbf{t}_i)$ and outputs $(\textsc{recv}, \mathsf{sid}, (i, \mathbf{x}_{r_i}^i))$, for each $i \in [\ell]$.

Figure 2: Semi-honest OT extension protocol $\Pi_{\mathsf{OT\text{-}e}}$

3 Our Solution

3.1 Semi-Honest Protocol

We will now explain our basic protocol that is secure in a semi-honest setting, where in addition $\mathcal{C}$ and $\mathcal{Q}$ are non-colluding. The parties involved are the data owners $\mathcal{P}_1, \ldots, \mathcal{P}_n$, where each $\mathcal{P}_i$ holds *persistent* input data $\mathbf{x}_i$ that is stored in the cloud $\mathcal{C}$, and $\mathcal{Q}$ who acts as the garbled circuit evaluator and holds input data $\mathbf{x}_\mathcal{Q}$. The parties anticipate that some subset $\{\mathcal{P}_i \mid i \in I\}$ of them will perform a cloud-assisted private computation with $\mathcal{Q}$ over their datasets at some later point in time.

In an offline phase, each party P_i samples $r_i \leftarrow \{0,1\}^\kappa$ uniformly at random, and uploads a secret-share $\mathbf{z}_i := \mathbf{x}_i \oplus g(r_i)$ of their dataset $\mathbf{x}_i$ to the cloud $\mathcal{C}$. Here $g(r_i)$ is an agreed-upon public PRF, such as AES in counter-mode keyed by r_i.

Let $I := \{I_1, \ldots, I_m\} \subseteq [n]$. At a later time, $\mathcal{Q}$ along with $\{\mathcal{P}_i \mid i \in I\}$ decide to evaluate a functionality

$$f(\{\mathbf{x}_i\}_{i \in I}, \mathbf{x}_\mathcal{Q}) := (f_1(\{\mathbf{x}_i\}_{i \in I}, \mathbf{x}_\mathcal{Q}), \ldots, \\ f_m(\{\mathbf{x}_i\}_{i \in I}, \mathbf{x}_\mathcal{Q}), f_\mathcal{Q}(\{\mathbf{x}_i\}_{i \in I}, \mathbf{x}_\mathcal{Q})) \ ,$$

where each data owner $\mathcal{P}_{I_j}$ learns $f_j(\{\mathbf{x}_i\}_{i \in I}, \mathbf{x}_\mathcal{Q})$, and $\mathcal{Q}$ learns $f_\mathcal{Q}(\{\mathbf{x}_i\}_{i \in I}, \mathbf{x}_\mathcal{Q})$. Any additional *per computation* input data $\mathbf{x}_i'$ for party $\mathcal{P}_i$ is expressed as being appended to the end of $\mathbf{z}_i$, and will be discussed in greater detail in 3.3.3. The cloud $\mathcal{C}$ verifies that all involved parties wish to compute f. The data owners $\{\mathcal{P}_i \mid i \in I\}$ send their secret seeds r_i to $\mathcal{Q}$, who computes the masks $g(r_i)$. A two-party secure computation is then performed between $\mathcal{C}$ and $\mathcal{Q}$ to compute the related functionality

$$\widetilde{f}\,(\{\mathbf{z}_i\}_{i \in I}, \{g(r_i)\}_{i \in I}, \mathbf{x}_\mathcal{Q}) := f\,(\{\mathbf{z}_i \oplus g(r_i)\}_{i \in I}, \mathbf{x}_\mathcal{Q}) \ .$$

To evaluate $\widetilde{f}$ securely using MPC, the cloud $\mathcal{C}$ acts as the garbler and generates the garbled circuit that computes the functionality $\widetilde{f}$, and sends it to $\mathcal{Q}$ (recall 2.1). In the oblivious transfer phase $\mathcal{Q}$ will select the input wire labels corresponding to $g(r_i)$. In the traditional Yao's protocol $\mathcal{C}$

would input the corresponding wire labels by their truth values resulting in $\mathcal{Q}$ obtaining the labels that encode $g(r_i)$. $\mathcal{C}$ would then send the labels encoding $\mathbf{z}_i$ so that $\mathcal{Q}$ could evaluate the $\widetilde{f}$ circuit. Instead, we employ an optimization where $\mathcal{C}$ inputs the wire labels for $g(r_i)$ into the OT protocol after permuting them by $\mathbf{z}_i$. This results in $\mathcal{Q}$ obtaining the effective input wire labels with values $\mathbf{x}_i = \mathbf{z}_i \oplus g(r_i)$ with no additional overhead. In particular, $\mathcal{C}$ only garbled the circuit corresponding to f and $\mathcal{Q}$ obliviously learns the wire labels encoding $\mathbf{x}_i$. After evaluating the garbled circuit, $\mathcal{Q}$ sends to party $\mathcal{P}_{I_j}$ the encoding information $\mathbf{y}_j$ (i.e. the *permute bits*) for the garbled output corresponding to the function f_j, and keeps the encoding information $\mathbf{y}_\mathcal{Q}$ corresponding to the garbled output of $f_\mathcal{Q}$ to itself. The cloud $\mathcal{C}$ sends $\mathcal{P}_{I_j}$ the corresponding decoding information $\mathbf{d}_j$ that $\mathcal{P}_{I_j}$ uses to obtain its result $f_j(\{\mathbf{x}_i\}_{i\in I}, \mathbf{x}_\mathcal{Q}) = \mathbf{d}_j \oplus \mathbf{y}_j$, and it sends $\mathcal{Q}$ the decoding information $\mathbf{d}_\mathcal{Q}$ that $\mathcal{Q}$ similarly uses to obtain its result $f_\mathcal{Q}(\{\mathbf{x}_i\}_{i\in I}, \mathbf{x}_\mathcal{Q}) = \mathbf{d}_\mathcal{Q} \oplus \mathbf{y}_\mathcal{Q}$.

This basic protocol securely and privately computes the functionality $f(\{\mathbf{x}_i\}_{i\in I}, \mathbf{x}_\mathcal{Q})$ under the assumption that the parties are semi-honest, and that $\mathcal{C}$ and $\mathcal{Q}$ are non-colluding. By the security properties of garbled circuits, $\mathcal{Q}$'s view of the output encoding information $\mathbf{y}_j$ (resp. $\mathbf{y}_\mathcal{Q}$) is uniformly distributed without the decoding information $\mathbf{d}_j$ (resp. $\mathbf{d}_\mathcal{Q}$). Therefore, the evaluator $\mathcal{Q}$ learns nothing more than their prescribed output and the r_i values that data stored in the cloud is encrypted under.

3.2 Enhanced Protocol

We will now describe several attacks against the semi-honest protocol and their respective solutions. Looking forward, we will arrive at a malicious secure protocol subject to a non-collusion assumption between the cloud and circuit evaluator.

3.2.1 Input Consistency

In 3.1 the party denoted by $\mathcal{Q}$ evaluates the garbled circuit computing the function f', which reconstructs the 2-out-of-2 secret shares of the logical inputs and subsequently evaluates f. This leads to a situation where $\mathcal{Q}$ can easily flip any set of input bits. In order to obtain malicious security, it is necessary for $\mathcal{Q}$ to prove that they provided the correct value for the input secret shares $g(r_i)$.

Suppose that instead of secret sharing $\mathcal{P}_i$'s input $\mathbf{x}_i$ between $\mathcal{C}$ and $\mathcal{Q}$, $\mathcal{P}_i$ simply performs oblivious transfer with $\mathcal{C}$ in the setup phase and forwards the wire labels to $\mathcal{Q}$ at the start of each computation. While this achieves the desired security, it leads to a situation where $\mathcal{P}_i$ must send a significant amount of data for each computation on its data. Instead, we will show how to adapt OT-e (recall 2.2) in a non-black box manner to also achieve persistent cloud storage for $\mathcal{P}_i$, as in 3.1, with minimal online interaction.

For simplicity, suppose there is only one data owner $\mathcal{P}$ who has a secret seed r that it has sent to $\mathcal{Q}$. In the next step $\mathcal{Q}$ needs to obtain the wire labels from $\mathcal{C}$ corresponding to the bits of the selection string $\mathbf{c} := g(r)$. Let ℓ denote the bit-length of $\mathbf{c}$. We now explain the protocol $\Pi_{\textsf{OT-e}}$ (see Figure 2) in the context of our setup. At a high level, OT-e works in three phases. First κ so called base OTs on κ-bit strings are performed. We note that these OTs are in the *reversed* direction relative to the final OT messages. That is, the cloud $\mathcal{C}$ acts as a receiver and $\mathcal{Q}$ acts as the sender with uniform messages $\mathbf{k}_0^i, \mathbf{k}_1^i \in \{0,1\}^\kappa$ in the i-th base OT. The cloud $\mathcal{C}$ samples $\Delta \in \{0,1\}^\kappa$ uniformly, and selects $\mathbf{k}_{\Delta_i}^i$.

In the second phase, suppose that OT-e should result in ℓ OTs of length κ of pairs $\{(\mathbf{m}_0^i, \mathbf{m}_1^i)\}$, where the receiver $\mathcal{Q}$ wants to learn the messages indexed by the selection string $\mathbf{c} \in \{0,1\}^\ell$, i.e. $\mathbf{m}_{c_i}^i$ for every $i \in [\ell]$. Both parties expand the $\mathbf{k}_b^i$ values to be ℓ bits by computing $\mathbf{t}_b^i := g(\mathbf{k}_b^i)$. The cloud $\mathcal{C}$ now holds the larger messages $\mathbf{t}^i_{\Delta_i} \in \{0,1\}^\ell$. $\mathcal{Q}$ knows both $\mathbf{t}^i_0$ and $\mathbf{t}^i_1$, but does not know which one is held by $\mathcal{C}$. The OT-e receiver $\mathcal{Q}$ then computes $\mathbf{u}^i := \mathbf{t}^i_0 \oplus \mathbf{t}^i_1 \oplus \mathbf{c}$ for every $i \in [\kappa]$, and sends them to $\mathcal{C}$. This is the final message sent by $\mathcal{Q}$ in the protocol, and commits it to the choice of the selection string $\mathbf{c}$.

In the last phase, the cloud $\mathcal{C}$ computes a matrix $\mathbf{q} \in \{0,1\}^{\ell\times\kappa}$, where the i-th column is $\mathbf{q}^i := (\Delta_i \cdot \mathbf{u}^i) \oplus \mathbf{t}^i_{\Delta_i} = \mathbf{t}^i_0 \oplus (\Delta_i \cdot \mathbf{c})$. Let $\mathbf{t}_0 \in \{0,1\}^{\ell\times\kappa}$ be the matrix whose i-th column vector is $\mathbf{t}^i_0$. Let $\mathbf{t}_i$ denote the i-th row of $\mathbf{t}$. Then, by definition, the i-th row $\mathbf{q}_i$ of $\mathbf{q}$ is $\mathbf{q}_i = \mathbf{t}_i \oplus (\mathbf{c}_i \cdot \Delta)$, where $\mathbf{t}_i$ is the i-th row of $\mathbf{t}$. To see this, consider the case when $\mathbf{c}_i = 1$. Then in the j-th bit location of the i-th row of $\mathbf{q}$ there is an additional $(\mathbf{c}_i \cdot \Delta_j) = \Delta_j$ additive term, and similarly when $\mathbf{c}_i = 0$ there is no additional term. The cloud $\mathcal{C}$ then *one-time pad* encrypts the i-th message pair $\{(\mathbf{m}^i_0, \mathbf{m}^i_1)\}$ as $\mathbf{y}^i_0 := \mathbf{m}^i_0 \oplus H(i, \mathbf{q}_i)$ and $\mathbf{y}^i_1 := \mathbf{m}^i_1 \oplus H(i, \mathbf{q}_i \oplus \Delta)$, and sends them to the receiver. The receiver $\mathcal{Q}$ can then compute $\mathbf{m}^i_{\mathbf{c}_i} = \mathbf{y}^i_{\mathbf{c}_i} \oplus H(\mathbf{t}_i)$.

We now show how to efficiently distribute this basic OT-e protocol described in Figure 2.2 to the setting where $\mathcal{P}$ chooses which messages are learned in the OT, while allowing $\mathcal{Q}$ to be the oblivious receiver. Critical in this observation is that the selection string $\mathbf{c}$ is fixed by $\mathcal{P}$ in the first two phases described above, i.e. by choosing the base OT messages $\mathbf{k}^i_0$, $\mathbf{k}^i_1$, and the matrix $\mathbf{u}$ (with column vectors $\mathbf{u}^i$). Once the cloud $\mathcal{C}$ receives these protocol messages, the final OT messages that can be learned by the receiver are fixed.

Thus, in the offline phase, $\mathcal{P}$ will upload its data to the cloud as $\mathbf{z} := \mathbf{x} \oplus \mathbf{c}$. $\mathcal{P}$ will perform the first two phases of OT-e using the correct $\mathbf{c}$, and send the matrix $\mathbf{u}$ to $\mathcal{C}$. $\mathcal{C}$ will then learn the matrix $\mathbf{q}$ whose i-th row is $\mathbf{q}_i = \mathbf{t}_i \oplus (\mathbf{c}_i \cdot \Delta)$. In the online phase, $\mathcal{P}$ will send the seed r and the seed used to derive the base OT messages $\mathbf{k}^i_b$ to $\mathcal{Q}$, who can then regenerate $\mathbf{u}$, $\mathbf{c} = g(r)$, and complete the OT-e with $\mathcal{C}$. To prevent $\mathcal{Q}$ from cheating, $\mathcal{C}$ only needs to check that the matrix $\mathbf{u}$ sent by $\mathcal{Q}$ is the same as that sent earlier in the offline phase by $\mathcal{P}$.

As in the semi-honest protocol of 3.1, $\mathcal{C}$ will permute the input wire labels that $\mathcal{Q}$ will use to evaluate the circuit with by $\mathbf{z} = \mathbf{x} \oplus \mathbf{c}$. This results in $\mathcal{Q}$ obtaining the wire labels encoding $\mathbf{x} = \mathbf{z} \oplus \mathbf{c}$, while remaining oblivious to the value of $\mathbf{x}$.

However, the downside of this approach is that $\mathcal{C}$ will have to store the $\kappa \times \ell$-matrix $\mathbf{u}$, which is potentially huge. In addition, this places a heavy communication cost for the data owners $\mathcal{P}_i$, which we would like to avoid. The simplest solution is for $\mathcal{P}$ to locally compute the matrix $\mathbf{u}$ in the offline phase, and send $\mathcal{C}$ only a small commitment to it. In the online phase $\mathcal{Q}$ will regenerate the matrix and eventually send it to $\mathcal{C}$, at which point $\mathcal{C}$ can open the commitment and verify that the matrix $\mathcal{Q}$ sent is the correct one. This adds only a small κ-bit communication overhead for $\mathcal{P}$.

In the case of multiple data owners $\mathcal{P}_1, \ldots, \mathcal{P}_n$, the protocol described above is simply performed for each of them individually.

3.2.2 Output Fairness

After $\mathcal{Q}$ has evaluated the garbled circuit and obtained the garbled outputs $\mathbf{y}_i$ of all involved parties $\mathcal{P}_i$ (and its own garbled output $\mathbf{y}_\mathcal{Q}$), it needs to distribute $\mathbf{y}_i$ to $\mathcal{P}_i$, who then obtains the corresponding decoding information $\mathbf{d}_i$ from $\mathcal{C}$ to recover the actual output bits. If $\mathcal{C}$ sends to $\mathcal{P}_i$ the wire labels for both logical outputs for each output bit of $\mathcal{P}_i$'s output, and one of them is what $\mathcal{Q}$ sent to $\mathcal{P}_i$, then $\mathcal{P}_i$ can be sure that $\mathcal{Q}$ indeed evaluated the circuit correctly and handed $\mathcal{P}_i$ the correct output wire label, as $\mathcal{Q}$ will never be able to learn more than one of the two output labels for any output wires.

Since $\mathcal{C}$ would need to send $\mathcal{P}_i$ two wire labels for each output bit, there is a possibly significant communication cost involved in this, which we would like to reduce. One step in this direction is for $\mathcal{C}$ to construct the output wire labels corresponding to $\mathcal{P}_i$'s output of the garbled circuit from a

PRF with a seed r_i^{out}. Now $\mathcal{C}$ can send r_i^{out} to $\mathcal{P}_i$, who can expand the PRF and obtain the output wire labels and decode the output, significantly reducing the communication cost.

There is still a possibly significant communication cost involved in $\mathcal{Q}$ sending the the appropriate output wire labels to $\mathcal{P}_i$. This cost can be reduced by instead using the *point-and-permute* technique first presented in [4]. Essentially, the garbling scheme will ensure that the last bits of each pair of output labels are different, so that $\mathcal{Q}$ only needs to send these last bits to $\mathcal{P}_i$ (*select bits*), who only needs to receive from $\mathcal{C}$ the permutation that matches them with the correct logical output bits. The problem with simply using this approach is that it makes it very easy for $\mathcal{Q}$ to flip any of the bits of $\mathcal{P}_i$'s output. To prevent this, $\mathcal{Q}$ will compute the XOR of all of the wire labels corresponding to $\mathcal{P}_i$'s output, and send it to $\mathcal{P}_i$. Now $\mathcal{C}$ will send to $\mathcal{P}_i$ the seed for the PRF to compute the entire output wire labels as we explained above. $\mathcal{P}_i$ can then compute the XOR of the appropriate labels received from $\mathcal{C}$ for each of its output wires, and verify that it matches the XOR received from $\mathcal{Q}$. This way $\mathcal{P}_i$ can be sure that the output bits it gets from $\mathcal{Q}$ are indeed the correct ones. Once all data owners have confirmed that they received valid encoded outputs from $\mathcal{Q}$, the semi-honest $\mathcal{C}$ will distribute the decoding information, and otherwise abort the protocol execution, guaranteeing fairness.

The communication cost in the output distribution and decoding process for $\mathcal{P}_i$ is therefore κ bits of communication with $\mathcal{C}$ and $\kappa + |\mathbf{y}_i|$ bits of communication with $\mathcal{Q}$.

3.3 Improvements

3.3.1 Random Access

Typically in applications of garbled circuits circuits are garbled and evaluated at a significantly faster rate than they are sent to the network interface, making the network communication into a bottleneck. If the data owner and evaluator want to hide their access patterns to the private data, they actually need to touch all of it at all times, which quickly becomes infeasible.

In many cases, however, it is not necessary to hide such access patterns, or only a part of the access pattern needs to be hidden. We now explain how we can enable efficient random access to private data inside the MPC, significantly reducing the size of data that needs to be touched, and thus significantly reducing the size of the garbled circuit that needs to be communicated in the online phase of our protocol.

Let $\mathcal{P}$ be any one of the data owners. Recall that in 3.2.1 $\mathcal{P}$ sends a commitment to its matrix $\mathbf{u}$ to $\mathcal{C}$. This commitment is opened by $\mathcal{Q}$ in the online phase of the protocol, guaranteeing that $\mathcal{Q}$ uses the correct selection string in the OT-e protocol $\Pi_{\mathsf{OT\text{-}e}}$ (recall Figure 2). Instead of committing to the entire matrix $\mathbf{u}$, $\mathcal{P}$ can instead build a *Merkle tree* of commitments, where the leaves are commitments to the individual rows of $\mathbf{u}$. The j-th row of $\mathbf{u}$ corresponds to randomness associated with the effective OT for the j-th wire label, so when $\mathcal{C}$ forms the garbled circuit it asks $\mathcal{P}$ to provide a Merkle chain of commitments for all of the rows of the $\mathbf{u}$ matrix that the circuit needs to touch. $\mathcal{Q}$ sends the corresponding rows of the matrix $\mathbf{u}$ to $\mathcal{C}$, who opens the commitments, guaranteeing that $\mathcal{Q}$ uses the correct selection string. This allows for efficient random access to the data inside the MPC phase, reducing the communication overhead to quasi-linear in the amount of data touched. More generally, one can imagine not committing to individual lines of the matrix $\mathbf{u}$ in the leaves of the Merkle tree, but to larger *leaf blocks* corresponding to, say, the bits of a certain database entry, or a file.

This line of thinking leads us to also consider using a tree-like structure of keys for $\mathcal{P}$, rather than a single key such as the r whose expansion $g(r)$ was earlier used to encrypt the entire data $\mathbf{x}$. For example, suppose we divide the data $\mathbf{x}$ into blocks of a certain size, e.g. into individual entries in a database, groups of entries, or files. For each such leaf block $\mathcal{P}$ assigns a seed that is used by a PRF to encrypt it. $\mathcal{P}$ then builds a tree by encrypting two or more of the leaf block keys under a new node key. Two or more of the node keys are then again encrypted using a new higher level node key, and so on, until a tree of encryptions of keys is formed. This tree of keys for each $\mathcal{P}$ can be stored by $\mathcal{C}$ along with the data of $\mathcal{P}$. When $\mathcal{Q}$ and $\mathcal{C}$ want to touch certain parts of the data of $\mathcal{P}$ in a secure computation, $\mathcal{C}$ shares with $\mathcal{Q}$ an appropriate set of encrypted nodes of the tree, who obtains the corresponding keys from $\mathcal{P}$, and can recover the seeds it needs for the PRF. This way only a limited amount of key material has to be shared with $\mathcal{Q}$. Of course, the larger the leaf blocks and the nodes are the more efficient this is both in terms of communication between $\mathcal{C}$ and $\mathcal{Q}$, and between $\mathcal{P}$ and $\mathcal{Q}$. Using larger leaf blocks would typically cause $\mathcal{P}$ to share more extra key material, however, by organizing the tree of keys in a certain way, $\mathcal{P}$ can efficiently control this leakage and possibly optimize for the types of computations that $\mathcal{Q}$ is most likely to request.

3.3.2 Updating the Keys

Let $\mathcal{P}$ be one of the data owners. Since $\mathcal{P}$ ends up sharing its secret key r with each buyer, there should be an easy way for it to revoke the key r and change the data stored by $\mathcal{C}$ to use a new key r'. A simple way to do this would be for $\mathcal{P}$ to send $g(r) \oplus g(r')$ to $\mathcal{C}$, who simply computes $\mathbf{z}' := \mathbf{z} \oplus (g(r) \oplus g(r'))$ to update the encryption. Unfortunately, this means that $\mathcal{P}$ has to send a linear amount of data to $\mathcal{C}$, which might in many cases be impractical, especially if $\mathcal{P}$ wants to update the key regularly.

Using the tree of keys described in 3.3.1 we can efficiently instead revoke and update selected parts of the key material. All that $\mathcal{P}$ needs to do is to send $\mathcal{C}$ updates to the appropriate nodes of the tree of keys.

3.3.3 Append, Delete, Update

$\mathcal{P}$ can at any point append data to $\mathbf{x}$ (which is stored in encrypted form $\mathbf{z}$ by $\mathcal{C}$) by computing further pseudo-random bits from the PRF using either a seed r that was used before, or possibly a new seed, depending for example on whether the optimizations of 3.3.1 are used. If a seed that was used also earlier is used for the newly appended data, it is crucial that none of the earlier pseudo-random bits generated from that seed are reused, as this leaks a linear relation between the updated data.

If trees of commitments and keys are used (see 3.3.1), then $\mathcal{P}$ will need to expand these trees in an appropriate way, depending on the nature of the data. For example, if the data is expected to be used a lot together with some other earlier data, then $\mathcal{P}$ might want to consider appending it to the same branch of the trees. Obviously $\mathcal{P}$ might need to update the relevant meta-data stored by $\mathcal{C}$.

$\mathcal{P}$ can at any point request $\mathcal{C}$ to delete certain parts of the data. Since we assume that $\mathcal{C}$ follows the protocol, we assume that this operation will be appropriately carried out, and the relevant meta-data updated correctly. To update any piece of data, $\mathcal{P}$ must first ask $\mathcal{C}$ to delete it, and subsequently append the replacement data as was explained above. As soon as all of the meta-data is updated, the value has been replaced. A *per computation* input of a party $\mathcal{P}_i$ can be expressed as appending data to the end of $\mathbf{x}_i$, which is then deleted before the next computation.

4 Security

We now discuss the security of our protocol. To keep the technical details sufficiently clean, we focus on a special case of our protocol, which does not involve parties updating their data or computing on subsets of the data. Nevertheless, this special case captures the main mechanisms and security considerations.

In Figure 3 we define the ideal functionality that our protocol securely realizes. Importantly, the functionality explicitly captures a setting with in which cloud-stored data of the parties is used for several computations. Furthermore, the role of the evaluator $\mathcal{Q}$ is not fixed, and can change from computation to computation. Finally, the functionality is *fair* in the sense that either all parties receive their output, or else no party receives any output.

Parameters: Parties $\mathcal{P}_1, \ldots, \mathcal{P}_n, \mathcal{C}$.

Data Upload: On input $\mathbf{x}_i$ from party $\mathcal{P}_i$, store $\mathbf{x}_i$ internally. This kind of command can happen once per party $\mathcal{P}_i$.

Computation: On input $(f, I, \mathcal{Q})$ from $\mathcal{C}$, where

- f is a function with $|I|$ inputs and $|I| + 1$ outputs
- $I = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$ denotes a set of input parties
- $\mathcal{Q}$ denotes a designated evaluator party (who may be one of the $\mathcal{P}_i$'s or any other party),

compute $(\mathbf{y}_{i_1}, \ldots, \mathbf{y}_{i_k}, \mathbf{y}_{\mathcal{Q}}) := f(\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_k})$. Give message READY to the adversary. When the adversary responds with DELIVER, then give $\mathbf{y}_i$ to $\mathcal{P}_i$ and $\mathbf{y}_{\mathcal{Q}}$ to $\mathcal{Q}$. This kind of command can happen any number of times.

Figure 3: Ideal functionality for SDE.

Kamara et al. [14] define a notion of *non-cooperating* adversaries. Suppose two parties A and B are both corrupt, and in particular A may be malicious. Informally, party A is *non-cooperating with respect to party* B if party A does not use its malicious capability to send B any useful information. More formally, B should be able to simulate the communication between A and B in the protocol.

We then show the following:

Theorem 1 *Our protocol securely realizes the ideal functionality in Figure 3 provided that:*

- *Party $\mathcal{C}$ may be corrupted only in a semi-honest way, while any other parties may be corrupted in a malicious way.*
- *If an adversary controls a set $\mathcal{A}$ of colluding parties, and there is ever a computation whose evaluator is among $\mathcal{A}$, then the adversary $\mathcal{A}$ must be non-cooperative with respect to $\mathcal{C}$.*

In short, the protocol is secure as long as the cloud is semi-honest and no evaluator cooperates with the cloud. This holds even if the parties $\mathcal{P}_i$ are otherwise malicious (simultaneous with the cloud being semi-honest).

Proof 1 (Sketch) *There are a few cases to consider. First, suppose no corrupted party ever acts as evaluator ($\mathcal{Q}$). Then the theorem gives no restriction on non-cooperation. By symmetry suppose the corrupted parties are $\mathcal{P}_1, \ldots, \mathcal{P}_k$ and $\mathcal{C}$ (assuming $\mathcal{C}$ to be corrupt only helps the adversary, however we require it to follow the protocol). We describe the simulation:*

The first time $\mathcal{P}_i$'s input is used in a computation, $\mathcal{P}_i$ must send to $\mathcal{Q}$ the value r_i as well as is OT-e protocol tape. $\mathcal{Q}$ will abort if these values are not consistent with the OT-e protocol transcript or the commitment to the final message. Hence the simulator can extract $\mathbf{x}_i = g(r_i) \oplus \mathbf{z}_i$ as $\mathcal{P}_i$'s input.[5] *Provided neither $\mathcal{Q}$ nor $\mathcal{C}$ abort, the OT-e protocol is effectively run semi-honestly. Hence $\mathcal{Q}$ will indeed learn the OT outputs corresponding to choice bits $g(r_i)$, while the opposite OT outputs are pseudorandom. Furthermore, this holds for* every *computation involving $\mathcal{P}_i$'s data.*

All that is left is to simulate the messages from honest $\mathcal{Q}$ to the $\mathcal{P}_i$'s. These messages depend on the garbled circuit output wire labels corresponding to $\mathcal{P}_i$'s output of f. Since $\mathcal{C}$ is running Yao's protocol honestly, we have in particular that $\mathcal{Q}$ evaluates a garbled circuit in the support of the garbling procedure. In that case, the output wire labels leak no more information than the prescribed output of f, so the messages from $\mathcal{Q}$ can be simulated given just the ideal output of $\mathcal{P}_1, \ldots, \mathcal{P}_k$.

Now we discuss the other case, in which a corrupted party acts as evaluator. From a useful lemma shown in [14, Lemma 6.1], it suffices to prove that: (1) the protocol is secure when all parties are semi-honest and use independent *simulators; (2) the protocol is secure when $\mathcal{A}$ (as in the theorem statement) is malicious and all other parties are honest.*

Part (1) is relatively straight-forward and omitted in the interest of space. For part (2), we consider an honest cloud. Then the cloud honestly acts as in Yao's protocol. The more challenging case is when $\mathcal{Q}$ is corrupt.

Consider the OT-e protocols involved in a computation. When $\mathcal{P}_i$ is corrupt, we can conceptually combine $\mathcal{Q}$ and $\mathcal{P}_i$ into a single corrupt OT-e receiver, playing against an honest $\mathcal{C}$. Hence the simulator can extract an effective OT input for $\mathcal{P}_i$, as described above, and this OT input will be the same for all computations. When $\mathcal{P}_i$ is honest, the binding property of the commitment ensures that all OT-e protocol messages are as intended by $\mathcal{P}_i$. By the argument above, $\mathcal{Q}$ will receive only the OT outputs corresponding to the $g(r_i)$ desired by $\mathcal{P}_i$. The corresponding share $\mathbf{z}_i$ for an honest $\mathcal{P}_i$ is known only to the honest parties.

Then by the security of Yao's protocol against a malicious receiver, we have that the remainder of the interaction with honest $\mathcal{C}$ can be simulated knowing only the prescribed output of f.

5 Implementation

We demonstrate several applications which build on the Secure Data Exchange framework. Our implementation is of the semi-honest protocol described in section 3.1. We note that the malicious protocol requires a minimal amount of overhead, and is not expected to significantly affect the running times reported in this section.

5.1 Test Platform

All performance numbers reported were obtained on a single server with simulated network latency. The server contains two 36-core Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30 GHz, and 256 GB of RAM. We executed our prototype in two network settings: a LAN configuration with all parties in the same network with 0.2 ms round-trip latency and 400 Mbps throughput, and a WAN configuration with a simulated 95 ms round-trip latency and 40 Mbps throughput. The network latency was simulated with the Linux `tc` command. All experiments were performed with a computational security parameter of $\kappa = 128$. The times reported are an average over 10 trials with variance

[5] We can assume without loss of generality that the cloud artificially calls the random oracle on $\mathbf{z}_i$ at the time it is uploaded, to allow the simulator to learn $\mathbf{z}_i$; note that this will always happen since we assume a semi-honest cloud.

between 2.1% − 8.0% in the LAN setting, and 6.0% − 16% in the WAN setting with a trend of smaller variance as n (see below) becomes larger.

We instantiate the garbled circuits using the state-of-the-art *half-gates* construction of [33]. The implementation employs the hardware accelerated AES-ni instruction set and uses fixed-key AES as the gate-level cipher, as suggested by [5]. Since circuit garbling and evaluation is the major computation bottleneck, we have taken great care to streamline and optimize the execution pipeline. However, all circuits generated in the following examples were created using a custom compiler which does not optimize the circuit size. Works such as TinyGarble [29] have shown that circuit sizes can be reduced by upwards of 80% when optimizers are applied. We expect that such techniques when applied to this protocol framework would result in a corresponding speedup. The semi-honest OT-e protocol is an instantiation of [16], and does not implement the malicious secure input commitment phase. The base OTs are performed using the protocol of [6].

5.2 Anova Test

Analysis of Variance (Anova) [24] is a family of statistical tests that compares the similarity of populations. In particular, the test determines whether the means of the populations differ by a statistically significant amount. We consider the scenario where four hospitals perform a computation over their joint patient data. Each hospital holds n patent records, where each patient has received one of four possible treatments. The hospitals wish to engage in a computation where the means of the treatment outcomes are compared. Due to patient privacy concerns, the hospitals are unable to aggregate the data in the clear. Instead, the hospitals can upload their data and a small amount of summary statistics to the cloud using our SDE protocol. In particular, each hospital will locally pre-compute and upload the mean, standard deviation, and variance of the four sets of treatment outcomes which they hold. They then engage in an SDE, where the Anova statistic is computed.

Figure 4 contains the running times in seconds and the overall communication cost of the anova test between four hospitals, a total of $4n$ patent records. By leveraging a preprocessing phase, the size of the circuit scales logarithmically in the size of the input dataset. As previously mentioned, each data owner computes the mean, standard deviation, and variance for their data. These summary values are then input into the secure data exchange protocol and all participants are return the result identifying whether there is a statistically significant difference in the treatment outcomes.

Another motivating scenario for the Anova test was mentioned in 1.1, where hospitals wish to compare the quality of treatment outcomes with respect to each other. In this case, each hospital would input their outcomes as a single set of treatments. The secure computation would then be used to decide if any hospital has significantly better or worse outcomes without *calling out* any single hospital for providing less effective care.

5.3 Chi Squared Test

Another motivating scenario mentioned in 1.1 was evaluation of the quality of data being placed on the market. In 1.1 this was explained in the context of a pharmaceutical company wishing to purchase patient data for which their model does not accurately predict the treatment outcomes. The model held by the data buyer may have significant value and must therefore be kept private. Likewise, data sellers do not wish to disclose their data until after it has been purchased. The *Chi Squared test* is a commonly used statistical test that computes the probability that a sample population follows a given model. For example, in this case the buyer would be interested in data

n	# ANDs	# XORs	Time		Comm.
			LAN	WAN	
2^{12}	351	485	0.12	0.9	9
2^{16}	462	632	0.13	1.42	10
2^{20}	584	799	0.13	1.50	13
2^{24}	721	985	0.25	1.93	20

Figure 4: Running time (sec.) and communication (MB) overhead of the Anova statistical test for four datasets of size n. The # ANDs (resp. # XORs) report the number of thousands of AND (resp. XOR) gates in the circuit.

n	# ANDs	# XORs	Time		Comm.
			LAN	WAN	
2^{8}	4.73	6.43	0.78	5.6	0.15
2^{10}	21.5	28.2	4.02	22	0.64
2^{12}	91.2	124	17.9	97	2.8
2^{14}	398	543	71.2	420	12

Figure 5: Running time (sec.) and communication (MB) overhead of the Chi Squared statistical test for two datasets of size n. The # ANDs (resp. # XORs) report the number of millions of AND (resp. XOR) gates in the circuit.

that does not fit their current model according to the Chi Squared test.

Figure 5 reports the running times of this use-case, where the Chi Squared statistic is computed for two independent datasets of size n. The data buyer provides a linear model with 24-bit coefficients, and receives a 32-bit Chi Squared value describing how well their model predicts the given datasets. Unlike the previous application, no preprocessing is performed on the data.

5.4 Machine Learning

Machine Learning techniques are now commonly being applied to numerous applications ranging from health care to business insights. However, due to privacy concerns the amount of data available for learning algorithms is often a limiting factor in the ability to generate accurate models. For instance, network monitoring systems analyze network traffic to detect abnormal activities, such as denial of service attacks, against a server. To increase the effectiveness of their models, many technology companies may wish aggregate their data, but are prevented by the reluctance to reveal such detailed information. Another prevalent yet significantly different example of machine learning are Genome-Wide Association Studies (GWAS), where the genes of a species are mapped to associated traits. However, when applied to human subjects, individuals may prefer not to participate in such studies due to privacy concerns.

Such limitations can be overcome by performing the computation within the SDE framework. Figure 6 reports the running times of a machine learning algorithm known as *Ridge Regression*. In this application, three participants each hold a dataset consisting of n 25-tuples of 8-bit values. The algorithm performs one complete iteration over all $3n$ records, and performs a stochastic gradient descent step for each record in a random order. The output of the computation is a linear model with 32-bit coefficients, which predicts the 25-th feature as a function of the remaining 24 features.

n	# ANDs	# XORs	Time		Comm.
			LAN	WAN	
2^8	24.0	37.4	4.12	25.1	0.78
2^{12}	425	648	64.2	424.2	12.6
2^{16}	6,533	10,010	1,045	6,213	207.0
2^{20}	104,181	157,623	23,654	100,706	3,179.1

Figure 6: Running time (sec.) and communication (MB) overhead of the machine learning task for three datasets of size n. The # ANDs (resp. # XORs) report the number of millions of AND (resp. XOR) gates in the circuit.

References

[1] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, pages 535–548. ACM, 2013.

[2] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In *Advances in Cryptology–EUROCRYPT 2015*, pages 673–701. Springer, 2015.

[3] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology*, 23(2):281–343, 2010.

[4] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, pages 503–513. ACM, 1990.

[5] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy*, pages 478–492, Berkeley, California, USA, May 19–22, 2013. IEEE Computer Society Press.

[6] Mihir Bellare and Silvio Micali. Non-interactive oblivious transfer and applications. In *Advances in Cryptology–CRYPTO 89 Proceedings*, pages 547–557. Springer, 1989.

[7] Henry Carter, Charles Lever, and Patrick Traynor. Whitewash: Outsourcing garbled circuit generation for mobile devices. In *Proceedings of the 30th Annual Computer Security Applications Conference*, pages 266–275. ACM, 2014.

[8] Henry Carter, Benjamin Mood, Patrick Traynor, and Kevin R. B. Butler. Secure outsourced garbled circuit evaluation for mobile devices. In Samuel T. King, editor, *Proceedings of the 22th USENIX Security Symposium, Washington, DC, USA, August 14-16, 2013*, pages 289–304. USENIX Association, 2013.

[9] Henry Carter, Benjamin Mood, Patrick Traynor, and Kevin R. B. Butler. Outsourcing secure two-party computation as a black box. In Michael Reiter and David Naccache, editors, *CANS 15*, LNCS, pages 214–222, Marrakesh, Morocco, December 10–12, 2015. Springer, Heidelberg, Germany.

[10] Uri Feige, Joe Killian, and Moni Naor. A minimal model for secure computation. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, pages 554–563. ACM, 1994.

[11] Ben A Fisc, Binh Vo, Fernando Krell, Abishek Kumarasubramanian, Vladimir Kolesnikov, Tal Malkin, and Steven M Bellovin. Malicious-client security in blind seer: A scalable private dbms. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 395–410. IEEE, 2015.

[12] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology-CRYPTO 2003*, pages 145–161. Springer, 2003.

[13] Ari Juels and Burton S Kaliski Jr. Pors: Proofs of retrievability for large files. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 584–597. ACM, 2007.

[14] Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multi-party computation. *IACR Cryptology ePrint Archive*, 2011:272, 2011.

[15] Seny Kamara, Payman Mohassel, and Ben Riva. Salus: A system for server-aided secure function evaluation. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, pages 797–808. ACM, 2012.

[16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In *Advances in Cryptology–CRYPTO 2015*, pages 724–741. Springer, 2015.

[17] Mehmet S Kiraz and Berry Schoenmakers. An efficient protocol for fair secure two-party computation. In *Topics in Cryptology–CT-RSA 2008*, pages 88–105. Springer, 2008.

[18] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *Automata, Languages and Programming*, pages 486–498. Springer, 2008.

[19] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology-EUROCRYPT 2007*, pages 52–78. Springer, 2007.

[20] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[21] Dahlia Malkhi, Noam Nisan, Benny Pinkas, Yaron Sella, et al. Fairplay-secure two-party computation system. In *USENIX Security Symposium*, volume 4. San Diego, CA, USA, 2004.

[22] Payman Mohassel and Matthew Franklin. Efficiency tradeoffs for malicious two-party computation. In *Public Key Cryptography-PKC 2006*, pages 458–473. Springer, 2006.

[23] Benjamin Mood, Debayan Gupta, Kevin R. B. Butler, and Joan Feigenbaum. Reuse it or lose it: More efficient secure computation through reuse of encrypted values. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14*, pages 582–596, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press.

[24] Jim Morrison. Statistics for engineers: An introduction. Wiley, 2009.

[25] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 245–254. ACM, 1999.

[26] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *Advances in Cryptology–CRYPTO 2012*, pages 681–700. Springer, 2012.

[27] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos Keromytis, and Steve Bellovin. Blind seer: A scalable private dbms. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 359–374. IEEE, 2014.

[28] Michael O Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive*, 2005:187, 2005.

[29] E. M. Songhori, S. U. Hussain, A. R. Sadeghi, T. Schneider, and F. Koushanfar. Tinygarble: Highly compressed and scalable sequential garbled circuits. In *2015 IEEE Symposium on Security and Privacy*, pages 411–428, May 2015.

[30] David P Woodruff. Revisiting the efficiency of malicious two-party computation. In *Advances in Cryptology-EUROCRYPT 2007*, pages 79–96. Springer, 2007.

[31] Andrew Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.

[32] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole. In *Advances in Cryptology-EUROCRYPT 2015*, pages 220–250. Springer, 2015.

[33] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.