Secure Outsourced Computation in a Multi-tenant Cloud

Seny Kamara
Microsoft Research
senyk@microsoft.com

Mariana Raykova *
Columbia University
mariana@cs.columbia.edu

Abstract

We present a general-purpose protocol that enables a client to delegate the computation of any function to a cluster of n machines in such a way that no adversary that corrupts at most $n-1$ machines can recover any information about the client's input or output. The protocol makes black-box use of multi-party computation (MPC) and secret sharing and inherits the security properties of the underlying MPC protocol (i.e., passive vs. adaptive security and security in the presence of a semi-honest vs. malicious adversary).

Using this protocol, a client can securely delegate any computation to a multi-tenant cloud so long as the adversary is not co-located on at least one machine in the cloud. Alternatively, a client can use our protocol to securely delegate its computation to multiple multi-tenant clouds so long as the adversary is not co-located on at least one machine in one of the clouds.

1 Introduction

Cloud infrastructures can be roughly categorized as either public or private. In a public cloud, the infrastructure is owned and managed by a cloud provider and made available "as a service" to clients. The benefits of using a public cloud infrastructure include reliability, elasticity (i.e., computational resources can be increased quickly) and cost-savings. There are several reasons public clouds are so cost-effective but the most important one is the use of multi-tenancy, i.e., multi-plexing the virtual machines (VM) of several clients on the same physical machine.

Multi-tenancy, however, introduces a number security concerns since a client cannot control which virtual machines are co-located on the same physical machine. The security concerns over multi-tenancy are often cited as the main hurdle to the adoption of cloud computing and, in fact, recent work has shown that it is possible for an adversary to map the infrastructure of a cloud co-locate an adversarial VM on the same physical machine [**?**]. So far, the approaches considered to alleviate these concerns include improving the isolation of VMs through the hypervisor and, as proposed in [**?**], offering clients (at extra cost) the option of running their VMs on a single-tenant machine.

In this work, we propose a cryptographic approach to the problem. We present a protocol for delegated computation that executes a client's computation on several physical machines and guarantees the confidentiality of the client's input and output as long as at least one of the physical machines is not running an adversarial VM. If the physical machines used are in different clouds, then the protocols will provide security as long as one machine in one of the clouds is not running an adversarial VM.

*Work done while interning at Microsoft Research.

2 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. The output x of an algorithm $\mathcal{A}$ is denoted by $x \leftarrow \mathcal{A}$. We refer to the ith element of a vector $\mathbf{v}$ as either v_i or $\mathbf{v}[i]$. Throughout k will refer to the security parameter. A function $\nu : \mathbb{N} \to \mathbb{N}$ is negligible in k if for every positive 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, $f(k) \leq \nu(k)$.

Multi-party functionalities. An n-party randomized functionality is a function $f : \mathbb{N} \times (\{0,1\}^*)^n \times \{0,1\}^* \to \{0,1\}^*$, where the first input is the security parameter k, the second input is a vector of strings $\mathbf{x}$, the third input is a set of random coins and the output is a vector of strings. In the context of MPC, each party P_i holds an input x_i and wants to receive output y_i, where $\overline{y} \leftarrow f(k, \mathbf{x}; r)$ for $r \xleftarrow{\$} \{0,1\}^{\mathsf{poly}(k)}$. Throughout this work, we will omit the security parameter and the coins and simply write $\overline{y} \leftarrow f(\mathbf{x})$. A functionality is deterministic if it only takes the security parameter and the strings $\mathbf{x}$ as inputs 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. [**?**] Section 7.3) so in this work we focus on the former.

Secret sharing. A threshold secret sharing scheme consists of two polynomial-time algorithms $\Sigma = (\mathsf{Share}, \mathsf{Recover})$ such that Share takes as input a secret x from some input space, a number of shares $n \in \mathbb{N}$ and a threshold $t < n$ and outputs n shares $(x_1, \ldots, x_n)$; and $\mathsf{Recover}$ takes as input a set of t shares and outputs a secret x. Σ is correct if $\mathsf{Recover}$ returns x when it is given any subset of t shares of x. It is hiding if, given any $q < t$ shares, no adversary can learn any partial information about the secret x. The hiding property is formalized by requiring that there exist a simulator $\mathcal{S}$ such that for all secrets x in the input space, for all $n = \mathsf{poly}(k)$ and all $t \leq n$, $\mathcal{S}$ can generate n shares that are indistinguishable from "real" shares, i.e., generated using Share.

2.1 Secure Multi-Party Computation

In this section we present the standard ideal/real-world security definition for MPC [**?**], which compares the real-world execution of a protocol for computing an n-party function f to the ideal-world evaluation of f by a trusted party.

In MPC dishonest players are modeled by a single adversary that is allowed to corrupt a subset of the parties. This "monolithic" adversary captures the possibility of collusion between the cheating parties. One typically 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. Another distinction can be made as to how the adversary chooses which parties to corrupt. If the adversary 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.

In the setting of MPC with dishonest majorities and a malicious adversary, certain adversarial behavior cannot be prevented. In particular, dishonest workers can choose not to participate in the computation, can compute on arbitrary inputs, or abort the computation prematurely. As such we only consider security with abort.

Real-world. At the beginning of the real-world execution each player P_i receives its input x_i, while the adversary $\mathcal{A}$ receives a set $I \subset [n]$ of corrupted parties if he is static and receives the security parameter if he is dynamic. The real execution of Π between the players $P = (P_1, \ldots, P_n)$ and the adversary $\mathcal{A}$, denoted $\text{REAL}^{\mathsf{mpc}}_{\Pi,\mathcal{A},I}(k, \mathbf{x})$, consists of the outputs of the honest players and the outputs of $\mathcal{A}$ (which can be arbitrary functions of their views).

Ideal-world. In the ideal execution the parties interact with a trusted third party that evaluates f. As in the real-world execution, the ideal-world execution begins with each player receiving its input x_i and the adversary receiving the set of corrupted parties I. The honest parties send their input x_i to the trusted party while the corrupted parties send values x_i if $\mathcal{A}$ is semi-honest and arbitrary values x'_i if $\mathcal{A}$ is malicious.

Output delivery works as follows. If any party sends $\perp$, the trusted party aborts the execution and returns $\perp$ to all parties. Otherwise, it computes $\overline{y} \leftarrow f(\mathbf{x})$ and sends $\{y_i\}_{i \in I}$ to the adversary. The adversary can then decide to abort or continue the execution. If $\mathcal{A}$ chooses to abort, the trusted party sends $\perp$ to all the honest parties. If the adversary chooses to continue, the trusted party sends y_i to honest party P_i.

The ideal evaluation of f between players $P = (P_1, \ldots, P_n)$ and adversary $\mathcal{A}$, denoted $\text{IDEAL}^{\mathsf{mpc}}_{f,\mathcal{A},I}(k, \mathbf{x})$, consists of the outputs of the honest players and the outputs of $\mathcal{A}$ (which can be arbitrary functions of their views).

Security. Roughly speaking, a protocol Π that implements a function f is considered secure if it emulates, in the real-world, an evaluation of f in the ideal-world. This is formalized by requiring that any real-world adversary $\mathcal{A}$ can be simulated in the ideal-world evaluation.

Definition 2.1 (Security)**.** *Let f be an n-party functionality and Π be a protocol. We say that Π t-securely computes f if for all* PPT *adversaries $\mathcal{A}$, there exists a* PPT *simulator $\mathcal{S}$ such that for all $I \subset [n]$ such that $|I| \leq t$,*

$$\left\{\text{REAL}^{\mathsf{mpc}}_{\Pi,\mathcal{A},I}(k, \mathbf{x})\right\}_{k \in \mathbb{N}, \mathbf{x} \in \{0,1\}^*} \overset{c}{\approx} \left\{\text{IDEAL}^{\mathsf{mpc}}_{f,\mathcal{S},I,\mathcal{L}}(k, \mathbf{x})\right\}_{k \in \mathbb{N}, \mathbf{x} \in \{0,1\}^*}.$$

If $\mathcal{A}$ is dynamic then it receives 1^k as input and chooses I during the execution.

2.2 Secure Delegated Computation

Secure delegation of computation allows a client to securely outsource the evaluation of a function f on a private input x to an untrusted cluster of workers. Roughly speaking, a secure delegation scheme should guarantee that (1) the workers will not learn any partial information about the client's input and output; and (2) that the function is computed correctly.

We formally capture these requirements in the ideal/real-world paradigm. Our definition is similar to the model for MPC with the exception that only one party (i.e., the client) provides inputs and receives outputs from the computation and that the adversary cannot corrupt the client. For completeness, we formally describe the model here.

Real-world. At the beginning of the real-world execution the client receives its input x while the workers have no input. If the adversary is static, it receives a set $I \subset [n]$ (where $|I| \leq t$) that designates the corrupted workers. If, on the other hand, $\mathcal{A}$ is dynamic then it will choose which workers to corrupt during the execution of the protocol. The real execution of Π between the client, the workers and the adversary $\mathcal{A}$ is denoted $\text{REAL}^{\mathsf{del}}_{\Pi,\mathcal{A},I}(k, x)$ and consists of the output of the client, the honest workers and $\mathcal{A}$ (which can be an arbitrary function of its view).

Ideal-world. In the ideal execution the parties interact with a trusted third party that evaluates the function f. As in the real-world execution, the ideal-world execution begins with the client receiving its input x. Again the workers receive no input. If the adversary is static, it receives a set $I \subset [n]$ (where $|I| \leq t$) that designates the corrupted machines, whereas if it is dynamic it chooses its corruptions during the execution.

The client sends its input x to the trusted party. If the adversary is malicious, the trusted party also asks $\mathcal{A}$ if it wishes to abort the computation. If so, the trusted party returns $\perp$ to the client and halts. If not, it computes and returns $y \leftarrow f(x)$ to the client.

The ideal evaluation of f between the client, the workers and the adversary, denoted $\text{IDEAL}^{\mathsf{del}}_{f,\mathcal{A},I}(k, \mathbf{x})$, consists of the outputs of the client, the honest workers and $\mathcal{A}$ (which can be arbitrary functions of their views).

Security. Roughly speaking, a delegation protocol Ω for a function f is considered secure if it emulates, in the real-world, an evaluation of f in the ideal-world. This is formalized by requiring that any real-world adversary $\mathcal{A}$ can be simulated in the ideal-world evaluation.

Definition 2.2 (Secure delegation)**.** *Let f be a function and Ω be a delegation protocol. We say that Ω t-securely computes f if for all* PPT *adversaries $\mathcal{A}$, there exists a* PPT *simulator $\mathcal{S}$ such that for all $I \subseteq [n]$ such that $|I| \leq t$,*

$$\left\{\text{REAL}^{\mathsf{del}}_{\Omega,\mathcal{A},I}\big(k,x\big)\right\}_{k\in\mathbb{N},x\in\{0,1\}^*} \overset{c}{\approx} \left\{\text{IDEAL}^{\mathsf{del}}_{f,\mathcal{S},I}\big(k,x\big)\right\}_{k\in\mathbb{N},x\in\{0,1\}^*}.$$

3 A General-Purpose Protocol

We now present a protocol Ω for secure delegation to a multi-tenant cloud. The protocol makes use of a MPC protocol Π and a secret sharing scheme $\Sigma = (\mathsf{Share}, \mathsf{Recover})$ and works as follows. The input x is first split into shares $(x_1, \ldots, x_n)$ and each share is sent to a worker. The workers then execute Π to securely evaluate a functionality f' defined as follows:

$$f'\big((x_1, r_1), \ldots, (x_n, r_n)\big) = \mathsf{Share}\big(f\big(\mathsf{Recover}(x_1, \ldots, x_n)\big), n; r_1 \oplus \cdots \oplus r_n\big).$$

The execution of Π will result with the workers each receiving a share of the output $y = f(x)$. After receiving these shares, the workers send them to the client who proceeds to recover the output y.

Intuitively, as long as at least one worker is honest, the adversary will not learn any information about either the input or the output of the client. The confidentiality of the input follows from the security of Π which guarantee that the corrupted workers will not learn any information about the honest workers' first inputs (i.e., their share of x) and from the security of Σ which guarantees that as long as at least one share is unknown to the adversary no information can be recovered about the secret (i.e., x). The confidentiality of the output follows from the security of Σ which guarantees that if at least one share remains unknown to the adversary then no information can be recovered about the output. This last property, however, only holds if the randomness used to generate the shares is uniform but this is guaranteed to hold if at least one worker is honest since $r_1 \oplus \cdots \oplus r_n$ is uniformly distributed as long as at least one r_i is.

We formalize this intuition in the following theorem whose proof will appear in the full version. Note that Ω inherits the security properties of the underlying MPC protocol but we only show it for security against adaptive and malicious adversaries.

Theorem 3.1. *If Π is t-secure against an adaptive and malicious adversary and Σ is a secure secret sharing scheme, then Ω, as described above, is secure against an adaptive and malicious adversary that corrupts at most t workers.*