Thesis for The Degree of Licentiate of Engineering

Outsourcing Computations to a Cloud That You Don't Trust

Georgia Tsaloli

CHALMERS

Division of Networks and Systems
Department of Computer Science & Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2019

Outsourcing Computations to a Cloud That You Don't Trust

Georgia Tsaloli

Technical Report No 201L
ISSN 1652-876X
Department of Computer Science & Engineering
Division of Networks and Systems
Chalmers University of Technology
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2019.

Abstract

In many application scenarios, data need to be collected, stored and processed. Often sensitive data are collected from IoT devices, which are constrained regarding their resources, and, thus, remote, untrusted cloud servers are required to perform the computations. However, cloud computing raises many security and privacy concerns since cloud providers cannot be fully trustworthy. Data owners want their sensitive information to remain private and expect confidentiality guarantees; while users want to utilize the computations' results and desire correctness guarantees. Furthermore, in some cases, standard cryptographic primitives are not sufficient to ensure that there is no leakage of information.

In this work, we focus on the problem of outsourcing joint computations from joint sensitive inputs to multiple untrusted servers, while at the same time achieving public verifiability (*i.e.,* everyone can verify the correctness of the computed result). Additionally, we investigate how to avoid any leakage of information by providing differential privacy guarantees on the outsourced computation. More precisely, we introduce the notion of verifiable homomorphic secret sharing (VHSS) which allows multiple clients to outsource joint computations on multiple servers providing also the capability to verify the correctness of the computed result. We propose a concrete instantiation of VHSS for the function that computes the product of n secret inputs. Besides, we suggest three instantiations of computing the sum of n secret inputs by employing homomorphic collision-resistant hash functions, linearly homomorphic signatures, and a threshold signature scheme, respectively. Moreover, we design a protocol that provides both differential privacy and verifiable computation guarantees for outsourced computations.

Keywords: function secret sharing, homomorphic secret sharing, verifiable computation, differential privacy, privacy-preservation, public verifiability

Acknowledgments

First, I would like to thank my supervisor Katerina, who gave me the opportunity to start this journey, believed in me and helped me think positively and learn a lot. Thank you for your trust!

All the people in my division, Networks and Systems, deserve a "thank you" for making work an enjoyable place. Thank you to all the administration people for being so nice and friendly and always giving their support :)

Thank you, Bei, for the helpful research discussions at the beginning of my Ph.D. studies. Thank you, Carlo, for always opening interesting "crypto" discussions trying to find ways for collaboration :)

Thank you to my colleagues for the nice moments we spend together, especially, in my favorite "after-work" gatherings.

Special thanks go to my family who made the person I am today and has been a great support for me all these years.

Last but not least, a big thank you goes to my friends (you know who you are) who have been there for me in negative situations but have also been the reason to smile (or to smile again) and to create happy memories. Thank you all for being in my life ♡

Georgia Tsaloli
Göteborg, September 2019

Contents

Introduction

Motivation

In modern society, many applications involve joint computations on data collected from multiple users. Computing statistics on electricity consumption via smart metering and on health-related information from patients' data are some instances of collecting data to obtain useful results. Since data are often collected from resource-constrained devices (IoT devices), that cannot support heavy computations, computationally powerful servers are required to store and process the collected information. However, often these servers are untrusted. It is possible that there are conflicting interests and, thus, the returned results of the computations might be incorrect. Furthermore, the collected data may concern the income of a person, his health information or even the profits of a company. This means that there is a need to ensure that confidential data remain private, even in the presence of untrusted parties, and, additionally, that only the computed results, in which all parties are interested in, are revealed rather than sensitive information of individuals. Of course, since the result of the computations is itself important, another challenge is to be able to certify that the computed result is correct and has not been altered by malicious parties.

Background

There exist different scenarios in the literature addressing the problem of outsourcing computations. In fact, a lot of work considers the *single-client, single-server* setting [3, 25, 31, 33, 39, 40, 44], where one client outsources a computation to a single server. Furthermore, some existing work focuses on the *multi-client* setting [32, 35] where the computations are outsourced by multiple clients as well as on the *multi-server* setting [2, 48], where the computations are performed by multiple servers. However, in many application scenarios (*e.g.,* collecting users' data from multiple sensors) multiple servers might be needed to perform the computations to avoid single points of failure. In this context, the *multi-client, multi-server* setting has received limited attention.

Many cryptographic primitives, such as verifiable computation, secret sharing, and differentially private mechanisms, are useful in this context, and also utilized in this thesis. Therefore, we provide a high-level introduction to these areas.

Secret Sharing Schemes

Given a set of parties $\{c_1, \ldots, c_m\}$, an access structure is a monotone[1] collection of non-empty subsets of the parties [7,8]. For a certain domain of secrets, a *secret sharing scheme* is a method such that a dealer (or data owner), assumed to be honest, breaks a secret x into shares $x_1, \ldots, x_m$ and distributes them to m parties in such a way that any unauthorized set of them learns nothing about x from their shares, yet any authorized set of the parties can reconstruct x [10, 24]. The first secret sharing schemes were designed by Shamir [42] and Blakley [12], and are based on polynomial interpolation and hyperplanes intersection respectively.

Consider the use case of cloud storage. Sensitive data owners use cloud services to store their confidential information. Yet, cloud providers cannot be fully trusted. Encryption of the data before uploading [23, 49] can partially solve this problem; however, problems such as availability failure or data loss in a single cloud service provider scenario are not prevented. Therefore, secret sharing allows cloud users to be able to control their confidential data and store them securely. In particular, data owners can separate their sensitive data into several sets (where each share reveals nothing about the confidential data) and store each share in different clouds (see Figure 1).

In a secret sharing scheme, a certain number of shareholders (or cloud

[1] A collection $\mathcal{A} \subseteq 2^{\{c_1,\ldots,c_m\}}$ is monotone if $B \in \mathcal{A}$ and $B \subseteq C$ imply that $C \in \mathcal{A}$.

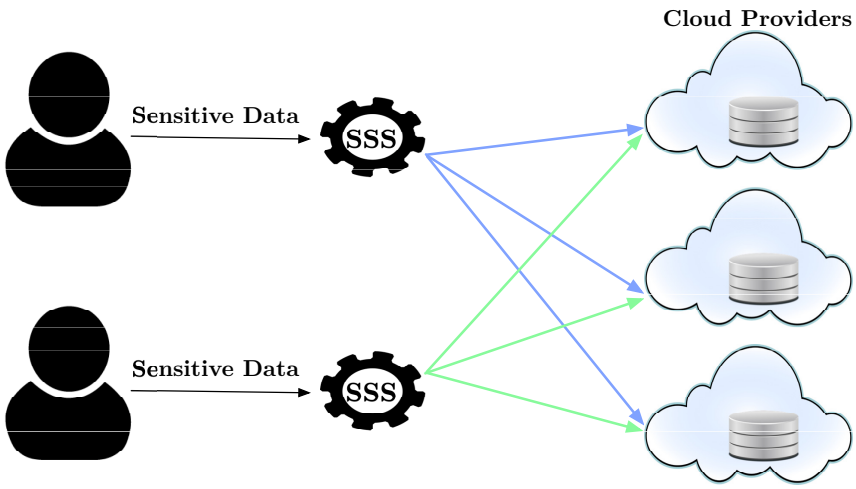

Figure 1: Sensitive data are split using a Secret Sharing Scheme (SSS) and distributed to multiple cloud service providers.

providers in the given example) is needed to reconstruct the sensitive data given from the data owners. Such a scheme can be employed to perform computations on the shared data. This fact often raises some security concerns about malicious behaviors of the shareholders. Verifiable computation techniques contribute to improving the security of this setting.

Verifiable Computation

Verifiable computation gives the ability to resource-constrained devices (*e.g.,* IoT devices), which outsource expensive computations to untrusted cloud providers, to verify the correctness of the returned results. The verification can be carried out either privately or publicly [1].

There are some essential requirements of *verifiable computation* [52]. More precisely, by essential requirements, we refer to requirements that any verifiable computation scheme must fulfill. These requirements are:

- **Verification Correctness**: This requirement means that it has to be almost impossible for an incorrect result to pass the test of a verifiable computation scheme. In other words, an honest verifier will confirm the correctness of the computation if and only if the result is correct. This is what we mean by security in verifiable computation mechanisms.

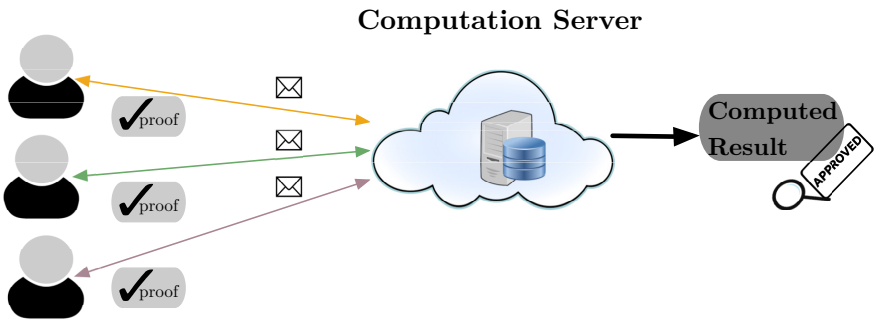

Figure 2: Users/Voters get proofs that their input/vote is included and everyone gets proof that the announced result is correct.

- **Verification Confidentiality**: This requirement is about the privacy of verifiable computation solutions. More precisely, it includes output privacy and, in some cases, input privacy. Input privacy refers to the fact that the servers should perform the computation but do not have access to the input data on which the computation is done. Output privacy means that the final result should be revealed in such a way that a verifier does not learn anything about the input to the computation.

- **High Verification Efficiency**: This requirement means that the verifiable computation scheme has to be practical. In other words, the verification process must require less time than performing the actual outsourced computation.

Several real-world scenarios are relevant to the concept of *verifiable computation*. Consider the case of electronic voting systems. Every voter wants to verify that their vote was collected and counted correctly. Additionally, everyone should be able to verify that the voting system is trustworthy and, therefore, the final result is true, while none of them needs to perform the computation itself (see Figure 2).

As one of the essential requirements stated, the final outsourced result must not reveal any information about the input data used in the computation. However, even when only the final result is revealed, some information can be inferred about the data of individual clients. *Differential privacy* deals with such privacy concerns.

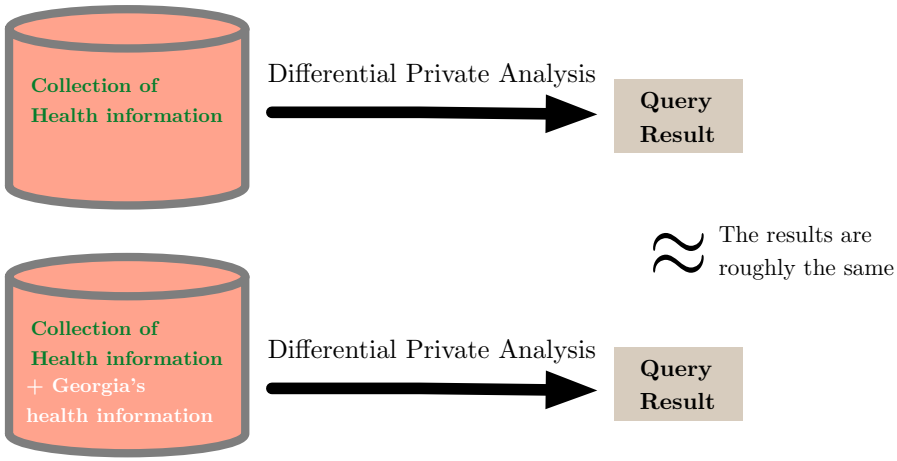

Figure 3: Georgia's health information is not leaked thanks to the differentially private analysis.

Differential Privacy

Differential privacy is a rigorous mathematical definition of privacy. The first formal definition of differential privacy was presented in [28]. *Differential privacy* makes it possible for companies and organizations to collect and share individuals' personal information while preserving their privacy. More specifically, *differential privacy* addresses the paradox of learning nothing about an individual while learning useful information about a population [29].

A differential private algorithm, which is applied to a dataset for performing a computation, has the following property: whether a single individual's record is inside the dataset or not, the output of the computation hardly changes (*i.e.,* the result is distorted by employing noise in the computed result). Consider, for instance, the case of computing statistics over a group of individuals about "how many people have the disease X"? A differential private algorithm guarantees that anyone having access to the result of this query will basically create the same conclusion whether or not any particular individual joined the input of the query (See Figure 3).

We should note here that, by increasing the noise added to the computed result, we provide high privacy guarantees. However, this can also lead to the low utility of the data. Therefore, it is important to maintain a good trade-off between privacy and utility when a differentially private mechanism is used.

Problem Statement and Research Questions

The need for outsourcing joint computations to a cloud arises from the fact that IoT devices, which collect the data, have typically limited resources and, therefore, cannot accommodate the heavy computations. Often, these computations are performed on data collected from multiple users (*e.g.,* computing electricity consumption statistics on data collected from multiple clients, monitoring environmental conditions from data collected from multiple sensors). Furthermore, in the cloud computing scenario, assigning the computation to a single server carries the risks of data loss or loss of confidentiality. This situation makes the *multi-client*, *multi-server* setting more realistic.

Malicious servers might want to learn sensitive information from the collected data about specific users. Hence, we need guarantees that users' confidential information is not leaked even though the computations are performed by the servers. Consequently, in this thesis, we want to address the following research question:

> **RQ1:** How can we outsource joint computations from sensitive datasets of multiple clients to multiple untrusted servers without revealing confidential data, while also avoiding single points of failure?

Another challenge while outsourcing computations to a cloud is that the servers might misbehave. More precisely, the servers, even if they do not have access to the raw data, might intentionally output an incorrect result. An incorrect computation's result is useless and, thus, it is essential that correctness guarantees are provided and that these guarantees are also publicly available. Retail companies, for example, need statistical results before announcing their offers and they need to know that these statistics are correct. Based on this challenge, the next research question that emerges is:

> **RQ2:** How can we provide public verifiability, in the context of outsourcing computations, while keeping our sensitive information private?

Another important concern is coming from the fact that companies want to aggregate big amounts of data to provide high-quality services to users (providing, for instance, personalized recommendations) but this comes as a risk of the users' privacy. In this scenario, traditional privacy-preserving tools (*e.g.,* encryption mechanisms) are not enough to encounter this issue. Individuals' data, even if not leaked, might be combined with auxiliary data and reveal sensitive information. Basic information of individuals might be

enough to narrow down the possible identity of a person [43] or even reveal their medical records [5]. The problem, then, is that we should be able to ensure that the returned result of the computation does not reveal more information than the result itself. Thus, the research question that comes out is:

> **RQ3:** How can we design a protocol which guarantees that the computed result does not leak information about the participation of an individual's data in the computation?

Contributions

We describe how we address the mentioned research questions in the context of the following research contributions.

Paper 1: Verifiable Homomorphic Secret Sharing [45]

This paper explores the problem of outsourcing joint computations in the *multi-client*, *multi-server* setting. More specifically, we introduce the notion of *verifiable homomorphic secret sharing* (VHSS) which allows n clients to outsource joint computations on their joint inputs to m servers without requiring any communication between the clients or the servers; providing, also, *public verifiability*, which means that every user has the capability to confirm that the final output of the computation is correct. The VHSS definition contributes to both **RQ1** and **RQ2**.

Furthermore, we provide a detailed example for casting Shamir's secret sharing scheme over a finite field $\mathbb{F}$ as an n-client, m-server, t-secure, additive HSS scheme for the function f which computes the sum of n field elements. This construction allows multiple clients to split their secret input x_i into shares which reveals nothing about their input and distribute them to multiple untrusted servers. The servers will output the sum of the secret inputs given. This solution contributes to the **RQ1** by protecting the clients' sensitive data, while also avoiding single points of failure by employing multiple servers for the computation.

Besides, we propose a concrete instantiation of an n-client, m-server, t-secure, multiplicative VHSS scheme. More precisely, this construction allows n clients to distribute shares of their secret inputs $x_1, \ldots, x_n$ to multiple servers. The latter output a function value $y = f(x_1, \ldots, x_n)$ which corresponds to the product of the n secret inputs as well as a proof σ that y is a correct. Anyone can verify that y is computed correctly. Therefore, we have

given a solution that contributes towards both **RQ1** and **RQ2** by outsourcing the joint computation of the multiplication function to multiple servers; while also achieving *public verifiability*.

Paper 2: Differential Privacy meets Verifiable Computation: Achieving Strong Privacy and Integrity Guarantees [46]

This short paper addresses the problem of outsourcing computations on sensitive datasets and publishing statistical results over a population of users while avoiding any leakage of information of individuals. In this paradigm, individuals want guarantees that their sensitive data will remain private and service providers want guarantees that the computation is correct. Since encryption mechanisms are not sufficient to avoid leakage of information, *differential privacy* can be employed to address this issue. Furthermore, *verifiable computation* mechanisms can be used to provide the verifiability property.

In this paper, we formalize the notion of verifiable differentially private computation and provide its formal definition. Subsequently, we propose a detailed protocol which provides both differential privacy and verifiable computation guarantees for the outsourced computations. We consider a curator which collects the data, an analyst, which performs the heavy computations, and a reader, who wants to get and verify the computed result, in our solution. We split our proposed protocol into two phases. The first phase is related to the differential privacy requirements and the second phase is related to the verifiable computation guarantees. More precisely, in the first phase, the reader and the curator agree on a randomness u to be used for the computation of the differential private function value. In the second phase, we employ our publicly verifiable differentially private computation scheme ($\mathsf{VDPC}_{\mathsf{Pub}}$) to achieve public verifiable differential privacy.

Hence, our solution contributes to the **RQ2** by providing *public verifiability* (*e.g.,* everyone can verify that the function result is a correct evaluation of the differentially private function). It also answers to the **RQ3** since the proposed protocol uses a differentially private algorithm for producing the computed result and, therefore, the participation of any individual's data is not revealed.

Paper 3: Sum it Up: Verifiable Additive Homomorphic Secret Sharing [47]

In this paper, we focus on the problem of outsourcing joint summations on joint inputs to external cloud servers. More precisely, we consider the

problem of *verifiable homomorphic secret sharing* (VHSS) for the function $f(x_1, \ldots, x_n) = x_1 + \ldots + x_n$ which computes the sum of n secret inputs. Our goal is to provide solutions such that: *(i)* clients protect their sensitive information from the untrusted cloud servers, *(ii)* servers perform computations in order to output the sum of the clients' data by holding shares of them, and, *(iii)* anyone is able to verify that the computed result is correct (*public verifiability*). We propose three concrete constructions to solve the *verifiable additive homomorphic secret sharing* (VAHSS) problem.

We employ three different methods to achieve (*public verifiability*) combined with an additive homomorphic secret sharing (additive HSS) scheme over a finite field $\mathbb{F}$ for computing the sum of the input data according to the requirements mentioned earlier. We discriminate three different cases not only depending on the primitive used for the verification but, also, depending on whether the partials proofs (values which are used for the generation of the proof of correctness) are generated by the clients or the servers. Furthermore, we slightly modify the VHSS definition [45] to capture the different cases regarding the generations of the proofs and, therefore, allow the use of VHSS in multiple application settings.

In our first VAHSS proposed instantiation, which is based on homomorphic collision-resistant hash functions, the clients are only responsible to split their secret data into shares and distribute them to the different servers; while the servers perform the computations needed for generating the final value y such that $y = x_1 + \ldots + x_n$; as well as a proof σ that y, indeed, corresponds to the sum of the secrets $x_1, \ldots, x_n$. In this scheme, the hash functions are used in the proof generation. Then, we suggest a construction based on linear homomorphic signatures. The clients distribute shares of their secrets to the servers and use their signing key to sign their secret data. The servers perform the calculations needed for the computation of $y = f(x_1, \ldots, x_n) = x_1 + \ldots + x_n$ as before. Finally, in our third construction, we employ a (t, n)-threshold RSA signature scheme which allows to successfully generate a proof σ even if $\mathsf{t} - 1$ servers are corrupted. In this setting, servers are responsible to compute both the function value y and the final proof σ that y is correct. The final proof is mutually produced by a coalition of t servers. In all three constructions, everyone can verify the correctness of the computation. Hence, with these three constructions, we contribute towards the **RQ1** by allowing multiple clients to outsource the sum computation to multiple untrusted servers without revealing their secret data; and towards the **RQ2** since we provide *public verifiability* in all proposed constructions.