A Pragmatic Introduction to Secure Multi-Party Computation

David Evans
University of Virginia
evans@virginia.edu

Vladimir Kolesnikov
Georgia Institute of Technology
kolesnikov@gatech.edu

Mike Rosulek
Oregon State University
rosulekm@eecs.oregonstate.edu

now
the essence of knowledge
Boston — Delft

A Pragmatic Introduction to Secure Multi-Party Computation

David Evans[1], Vladimir Kolesnikov[2] and Mike Rosulek[3]

[1] *University of Virginia; evans@virginia.edu*
[2] *Georgia Institute of Technology; kolesnikov@gatech.edu*
[3] *Oregon State University, rosulekm@eecs.oregonstate.edu*

ABSTRACT

Secure multi-party computation (MPC) has evolved from a theoretical curiosity in the 1980s to a tool for building real systems today. Over the past decade, MPC has been one of the most active research areas in both theoretical and applied cryptography. This book introduces several important MPC protocols, and surveys methods for improving the efficiency of privacy-preserving applications built using MPC. Besides giving a broad overview of the field and the insights of the main constructions, we overview the most currently active areas of MPC research and aim to give readers insights into what problems are practically solvable using MPC today and how different threat models and assumptions impact the practicality of different approaches.

David Evans, Vladimir Kolesnikov and Mike Rosulek, *A Pragmatic Introduction to Secure Multi-Party Computation*. NOW Publishers, 2018. (This version: April 15, 2020)

Contents

1

Introduction

Secure multi-party computation (MPC) enable a group to jointly perform a computation without disclosing any participant's private inputs. The participants agree on a function to compute, and then can use an MPC protocol to jointly compute the output of that function on their secret inputs without revealing them. Since its introduction by Andrew Yao in the 1980s, multi-party computation has developed from a theoretical curiosity to an important tool for building large-scale privacy-preserving applications.

This book provides an introduction to multi-party computation for practitioners interested in building privacy-preserving applications and researchers who want to work in the area. We provide an introduction to the foundations of MPC and describe the current state of the art. Our goal is to enable readers to understand what is possible today, and what may be possible in the future, and to provide a starting point for building applications using MPC and for developing MPC protocols, implementations, tools, and applications. As such, we focus on practical aspects, and do not provide formal proofs.

The term *secure computation* is used to broadly encompass all methods for performing computation on data while keeping that data secret. A computation method may also allow participants to confirm the result is indeed the output of the function on the provided inputs, which is known as *verifiable computation*.

There are two main types of secure and verifiable computation: *outsourced computation* and *multi-party computation*. Our focus is on multi-party computation, but first we briefly describe outsourced computation to distinguish it from multi-party computation.

1.1 Outsourced Computation

In an outsourced computation, one party owns the data and wants to be able to obtain the result of computation on that data. The second party receives and stores the data in an encrypted form, performs computation on the encrypted data, and provides the encrypted results to the data owner, without learning anything about the input data, intermediate values, or final result. The data owner can then decrypt the returned results to obtain the output.

Homomorphic encryption allows operations on encrypted data, and is a natural primitive to implement outsourced computation. With *partially-homomorphic encryption* schemes, only certain operations can be performed. Several efficient partially-homomorphic encryption schemes are known (Paillier, 1999; Naccache and Stern, 1998; Boneh *et al.*, 2005). Systems built on them are limited to specialized problems that can be framed in terms of the supported operations.

To provide *fully homomorphic encryption* (FHE), it is necessary to support a Turing-complete set of operations (e.g., both addition and multiplication) so that any function can be computed. Although the goal of FHE was envisioned by Rivest *et al.* (1978), it took more than 30 years before the first FHE scheme was proposed by Gentry (2009), building on lattice-based cryptography. Although there has been much recent interest in implementing FHE schemes Gentry and Halevi (2011), Halevi and Shoup (2015), and Chillotti *et al.* (2016), building secure, deployable, scalable systems using FHE remains an elusive goal.

In their basic forms, FHE and MPC address different aspects of MPC, and as such shouldn't be directly compared. They do, however, provide similar functionalities, and there are ways to adapt FHE to use multiple keys that enables multi-party computation using FHE (Asharov *et al.*, 2012; López-Alt *et al.*, 2012; Mukherjee and Wichs, 2016). FHE offers an asymptotic communication improvement in comparison with MPC, but at the expense of computational efficiency. State-of-the-art FHE implementations (Chillotti *et al.*, 2017) are thousands of times slower than two-party and multi-party

secure computation in typical applications and settings considered in literature. Ultimately, the relative performance of FHE and MPC depends on the relative costs of computation and bandwidth. For high-bandwidth settings, such as where devices connected within a data center, MPC vastly outperforms FHE. As FHE techniques improve, and the relative cost of bandwidth over computation increases, FHE-based techniques may eventually become competitive with MPC for many applications.

We do not specifically consider outsourcing computation or FHE further in this book, but note that some of the techniques developed to improve multi-party computation also apply to FHE and outsourcing. Shan *et al.* (2017) provide a survey of work in the area of outsourcing.

1.2 Multi-Party Computation

The goal of secure multi-party computation (MPC) is to enable a group of independent data owners who do not trust each other or any common third party to jointly compute a function that depends on all of their private inputs. MPC differs from outsourced computation in that all of the protocol participants are data owners who participate in executing a protocol. Chapter 2 provides a more formal definition of MPC, and introduces the most commonly considered threat models.

Brief history of MPC. The idea of secure computation was introduced by Andrew Yao in the early 1980s (Yao, 1982). That paper introduced a general notion of secure computation, in which m parties want to jointly compute a function $f(x_1, x_2, \ldots, x_m)$ where x_i is the i^{th} party's private input. In a series of talks over the next few years (but not included in any formal publication), Yao introduced the Garbled Circuits Protocol which we describe in detail in Section 3.1. This protocol remains the basis for many of the most efficient MPC implementations.

Secure computation was primarily of only theoretical interest for the next twenty years; it was not until the 2000s that algorithmic improvements and computing costs had reached a point where it became realistic to think about building practical systems using general-purpose multi-party computation. Fairplay (Malkhi *et al.*, 2004) was the first notable implementation of a general-purpose secure computation system. Fairplay demonstrated the possibility that

a privacy-preserving program could be expressed in a high level language and compiled to executables that could be run by the data-owning participants as a multi-party protocol. However, its scalability and performance limited its use to toy programs — the largest application reported in the Fairplay paper was computing the median two sorted arrays where each party's input is ten 16-bit numbers in sorted order, involving execution of 4383 gates and taking over 7 seconds to execute (with both parties connected over a LAN). Since then, the speed of MPC protocols has improved by more than five orders of magnitude due to a combination of cryptographic, protocol, network and hardware improvements. This enabled MPC applications to scale to a wide range of interesting and important applications.

Generic and specialized MPC. Yao's garbled circuits protocol is a *generic* protocol—it can be used to compute any discrete function that can be represented as a fixed-size circuit. One important sub-area of MPC focuses on specific functionalities, such as private set intersection (PSI). For specific functionalities, there may be custom protocols that are much more efficient than the best generic protocols. Specific functionalities can be interesting in their own right, but also can be natural building blocks for use in other applications. We focus mostly on generic MPC protocols, but include discussion of private set intersection (Section 3.8.1) as a particularly useful functionality.

1.3 MPC Applications

MPC enables privacy-preserving applications where multiple mutually distrusting data owners cooperate to compute a function. Here, we highlight a few illustrative examples of privacy-preserving applications that can be built using MPC. This list is far from exhaustive, and is meant merely to give an idea of the range and scale of MPC applications.

Yao's Millionaires Problem. The toy problem that was used to introduce secure computation is not meant as a useful application. Yao (1982) introduces it simply: "Two millionaires wish to know who is richer; however, they do not want to find out inadvertently any additional information about each other's wealth." That is, the goal is to compute the Boolean result of $x_1 \leq x_2$ where x_1 is the first party's private input and x_2 is the second party's private input.

Although it is a toy problem, Yao's Millionaires Problem can still be useful for illustrating issues in MPC applications.

Secure auctions. The need for privacy in auctions is well understood. Indeed, it is crucial for all participants, both bidders and sellers, to be able to rely on the privacy and non-malleability of bids. *Bid privacy* requires that no player may learn any other player's bid (other than perhaps revealing the winning bid upon the completion of the auction). *Bid non-malleability* means that a player's bid may not be manipulated to generate a related bid. For example, if a party generates a bid of $\$n$, then another party should not be able to use this bid to produce a bid of $\$n + 1$. Note that bid privacy does not necessarily imply bid non-malleability — indeed it is possible to design auction protocols that would hide a bid of $\$n$ while still allowing others to generate a related bid $\$n + 1$.

These properties are crucial in many standard bidding processes. For example, a sealed bid auction is an auction where bidders submit private (sealed) bids in attempts to purchase property, selling to the highest bidder. Clearly, the first bidder's bid value must be kept secret from other potential bidders to prevent those bidders from having an unfair advantage. Similarly, bid malleability may allow a dishonest bidder Bob to present a bid just slightly over Alice's bid, again, gaining an unfair advantage. Finally, the auction itself must be conducted correctly, awarding the item to the highest bidder for the amount of their bid.

A Vickrey auction is a type of sealed-bid auction where instead paying the value of their own bid, the highest bidder wins but the price paid is the value of the second-highest bid. This type of auction gives bidders an incentive to bid their true value, but requires privacy and non-malleability of each bid, and correctness in determining the winner and price.

MPC can be used to easily achieve all these features since it is only necessary to embed the desired properties into the function used to jointly execute the auction. All the participants can verify the function and then rely on the MPC protocol to provide high confidence that the auction will be conducted confidentially and fairly.

Voting. Secure electronic voting, in a simple form, is simply computation of the addition function which tallies the vote. Privacy and non-malleability of the vote (properties discussed above in the context of auctions) are essential for similar technical reasons. Additionally, because voting is a fundamental civil process, these properties are often asserted by legislation.

As a side note, we remark that voting is an example of an application which may require properties *not covered* by the standard MPC security definitions. In particular, the property of *coercion resistance* is not standard in MPC (but can be formally expressed and achieved (Küsters *et al.*, 2012)). The issue here is the ability of voters to *prove* to a third party how they voted. If such a proof is possible (e.g., a proof might exhibit the randomness used in generating the vote, which the adversary may have seen), then voter coercion is also possible. We don't delve into the specific aspects of secure voting beyond listing it here as a natural application of MPC.

Secure machine learning. MPC can be used to enable privacy in both the inference and training phases of machine learning systems.

Oblivious model inference allows a client to submit a request to a server holding a pre-trained model, keeping the request private from the server $\mathcal{S}$ and the model private from the client C. In this setting, the inputs to the MPC are the private model from $\mathcal{S}$, and the private test input from C, and the output (decoded only for C) is the model's prediction. An example of recent work in this setting include MiniONN (Liu *et al.*, 2017), which provided a mechanism for allowing any standard neural network to be converted to an oblivious model service using a combination of MPC and homomorphic encryption techniques.

In the training phase, MPC can be used to enable a group of parties to train a model based on their combined data without exposing that data. For the large scale data sets needed for most machine learning applications, it is not feasible to perform training across private data sets as a generic many-party computation. Instead, hybrid approaches have been designed that combine MPC with homomorphic encryption (Nikolaenko *et al.*, 2013b; Gascón *et al.*, 2017) or develop custom protocols to perform secure arithmetic operations efficiently (Mohassel and Zhang, 2017). These approaches can scale to data sets containing many millions of elements.

Other applications. Many other interesting applications have been proposed for using MPC to enable privacy. A few examples include privacy-preserving network security monitoring (Burkhart *et al.*, 2010), privacy-preserving genomics (Wang *et al.*, 2015a; Jagadeesh *et al.*, 2017), private stable matching (Doerner *et al.*, 2016), contact discovery (Li *et al.*, 2013; De Cristofaro *et al.*, 2013), ad conversion (Kreuter, 2017), and spam filtering on encrypted email (Gupta *et al.*, 2017).

1.3.1 Deployments

Although MPC has seen much success as a research area and in experimental use, we are still in the early stages of deploying MPC solutions to real problems. Successful deployment of an MPC protocol to solve a problem involving independent and mutually distrusting data owners requires addressing a number of challenging problems beyond the MPC execution itself. Examples of these problems include building confidence in the system that will execute the protocol, understanding what sensitive information might be inferred from the revealed output of the MPC, and enabling decision makers charged with protecting sensitive data but without technical cryptography background to understand the security implications of participating in the MPC.

Despite these challenges, there have been several successful deployments of MPC and a number of companies now focus on providing MPC-based solutions. We emphasize that in this early stage of MPC penetration and awareness, MPC is primarily deployed as an *enabler* of data sharing. In other words, organizations are typically not seeking to use MPC to add a layer of privacy in an otherwise viable application (we believe this is yet forthcoming). Rather, MPC is used to enable a feature or an entire application, which otherwise would not be possible (or would require trust in specialized hardware), due to the value of the shared data, protective privacy legislation, or mistrust of the participants.

Danish sugar beets auction. In what is widely considered to be the first commercial application of MPC, Danish researchers collaborated with the Danish government and stakeholders to create an auction and bidding platform for sugar beet production contracts. As reported in Bogetoft *et al.* (2009), bid privacy and auction security were seen as essential for auction participants.

The farmers felt that their bids reflected their capabilities and costs, which they did not want to reveal to Danisco, the only company in Denmark that processed sugar beets. At the same time, Danisco needed to be involved in the auction as the contracts were securities directly affecting the company.

The auction was implemented as a three-party MPC among representatives for Danisco, the farmer's association (DKS) and the researchers (SIMAP project). As explained by Bogetoft *et al.* (2009), a three party solution was selected, partly because it was natural in the given scenario, but also because it allowed using efficient information theoretic tools such as secret sharing. The project led to the formation of a company, Partisia, that uses MPC to support auctions for industries such as spectrum and energy markets, as well as related applications such as data exchange (Gallagher *et al.*, 2017).

Estonian students study. In Estonia, a country with arguably the most advanced e-government and technology awareness, alarms were raised about graduation rates of IT students. Surprisingly, in 2012, nearly 43% of IT students enrolled in the previous five years had failed to graduate. One potential explanation considered was that the IT industry was hiring too aggressively, luring students away from completing their studies. The Estonian Association of Information and Communication Technology wanted to investigate by mining education and tax records to see if there was a correlation. However, privacy legislation prevented data sharing across the Ministry of Education and the Tax Board. In fact, k-anonymity-based sharing was allowed, but it would have resulted in low-quality analysis, since many students would not have had sufficiently large groups of peers with similar qualities.

MPC provided a solution, facilitated by the Estonian company Cybernetica using their Sharemind framework (Bogdanov *et al.*, 2008a). The data analysis was done as a three-party computation, with servers representing the Estonian Information System's Authority, the Ministry of Finance, and Cybernetica. The study, reported in Cybernetica (2015) and Bogdanov (2015), found that there was no correlation between working during studies and failure to graduate on time, but that more education was correlated with higher income.

Boston wage equity study. An initiative of the City of Boston and the Boston Women's Workforce Council (BWWC) aims to identify salary inequities

across various employee gender and ethnic demographics at different levels of employment, from executive to entry-level positions. This initiative is widely supported by the Boston area organizations, but privacy concerns prevented direct sharing of salary data. In response, Boston University researchers designed and implemented a web-based MPC aggregation tool, which allowed employers to submit the salary data privately and with full technical and legal protection, for the purposes of the study.

As reported by Bestavros *et al.* (2017), MPC enabled the BWWC to conduct their analysis and produce a report presenting their findings. The effort included a series of meetings with stakeholders to convey the risks and benefits of participating in the MPC, and considered the importance of addressing usability and trust concerns. One indirect result of this work is inclusion of secure multi-party computation as a requirement in a bill for student data analysis recently introduced in the United States Senate (Wyden, 2017).

Key management. One of the biggest problems faced by organizations today is safeguarding sensitive data as it is being used. This is best illustrated using the example of authentication keys. This use case lies at the core of the product offering of Unbound Tech (Unbound Tech, 2018). Unlike other uses of MPC where the goal is to protect data owned by multiple parties from exposure, here the goal is to protect from compromise the data owned by a single entity.

To enable a secure login facility, an organization must maintain private keys. Let's consider the example of shared-key authentication, where each user has shared a randomly chosen secret key with the organization. Each time the user U authenticates, the organization's server S looks up the database of keys and retrieves U's public key sk_U, which is then used to authenticate and admit U to the network by running key exchange.

The security community has long accepted that it is nearly impossible to operate a fully secure complex system, and an adversary will be able to penetrate and stealthily take control over some of the network nodes. Such an advanced adversary, sometimes called Advanced Persistent Threat (APT), aims to quietly undermine the organization. Naturally, the most prized target for APT and other types of attackers is the key server.

MPC can play a significant role in *hardening* the key server by splitting its functionality into two (or more) hosts, say, S_1 and S_2, and secret-sharing

key material among the two servers. Now, an attacker must compromise *both* S_1 and S_2 to gain access to the keys. We can run S_1 and S_2 on two different software stacks to minimize the chance that they will both be vulnerable to the exploit available to the malware, and operate them using two different sub-organizations to minimize insider threats. Of course, routine execution does need access to the keys to provide authentication service; at the same time, key should never be reconstructed as the reconstructing party will be the target of the APT attack. Instead, the three players, S_1, S_2, and the authenticating user U, will run the authentication inside MPC, *without ever reconstructing any secrets*, thus removing the singular vulnerability and hardening the defense.

1.4 Overview

Because MPC is a vibrant and active research area, it is possible to cover only a small fraction of the most important work in this book. We mainly discuss generic MPC techniques, focusing mostly on the two-party scenario, and emphasizing a setting where all but one of the parties may be corrupted. In the next chapter, we provide a formal definition of secure multi-party computation and introduce security models that are widely-used in MPC. Although we do not include formal security proofs in this book, it is essential to have clear definitions to understand the specific guarantees that MPC provides. Chapter 3 describes several fundamental MPC protocols, focusing on the most widely-used protocols that resist any number of corruptions. Chapter 4 surveys techniques that have been developed to enable efficient implementations of MPC protocols, and Chapter 5 describes methods that have been used to provide sub-linear memory abstractions for MPC.

Chapters 3–5 target the weak semi-honest adversary model for MPC (defined in Chapter 2), in which is it assumed that all parties follow the protocol as specified. In Chapter 6, we consider how MPC protocols can be hardened to provide security against active adversaries, and Chapter 7 explores some alternative threat models that enable trade-offs between security and efficiency. We conclude in Chapter 8, outlining the trajectory of MPC research and practice, and suggesting possible directions for the future.