Course Introduction

CS 507, Topics in Cryptography: Secure Computation

David Heath

Fall 2026

This course studies secure multiparty computation—abbreviated MPC, or sometimes SMC by those outside the community—and related advanced topics. MPC is a powerful subfield of cryptography that, roughly, allows p mutually untrusting parties $P_0, \ldots, P_{p-1}$ to:

1. encode their respective private data $x_0, \ldots, x_{p-1}$,
2. collect together the encoded data,
3. run an agreed-upon program f on the collected data, yielding encoded output $y = f(x_0, \ldots, x_{p-1})$, and
4. decode the output y.

The key point is that while the parties $P_0, \ldots, P_{p-1}$ learn the program output y, they do not learn the program inputs. In other words:

> MPC allows mutually untrusting parties to run programs on their joint private inputs while learning nothing beyond (1) their own input and (2) the program output.

This capability is achieved by means of a cryptographic protocol, and such protocols can be used as a basis for privacy-preserving computations.

A Toy Example

The above description is abstract, and can be confusing. Let's examine a (slight generalization of a) classic, albeit contrived, example, first proposed by Andrew Yao, one of the pioneers of this field [Yao82].

Problem 1 (Yao's Millionaire's Problem – informal)**.** *Consider p millionaires who all attend an expensive dinner. Each millionaire P_i has a total wealth of x_i US dollars. (For simplicity, assume all x_i are distinct.) The millionaires now must pay the dinner bill, and they would like to arrange that the richest individual among them will pay. However, these millionaires are jealous, and each of them wishes to hide how much money they have. Thus, the millionaires wish to identify the millionaire P_i who has the most money, without any one of them revealing their individual wealth. In other words, the parties wish to compute the following function* ***securely****:*

$$\arg\max_i x_i.$$

When approached naively, this problem has no obvious solution. It seems intuitive that in order to compute the identity of the richest millionaire, we must collect information about the millionaires' wealth in such a way that some party will learn extra information.

Amazingly, this problem is, in fact, solvable. Indeed, it can be solved by applying the cryptographic techniques we will study in this course. Even more amazingly, those same techniques will allow our p parties not just to learn the simple arg max function of their inputs, but rather to learn any computable function of their choice of their inputs.

Why Study MPC?

MPC is a fascinating area of study that establishes strong connections between the theoretical and applied sides of cryptography, and of computer science.

On the theory side, we will have opportunities to explore new upper- and lower-bounds. We will see interesting and novel data structures, algorithms, and protocols. And studying this area provides a vehicle by which to study other related topics in cryptography, theoretical computer science, algorithms, and security. For example, we will touch on secret sharing, zero knowledge proofs, oblivious RAM, relationships between models of computation, function secret sharing, and much more.

On the applied side, MPC provides potential to address real-world problems that require privacy-preserving solutions. Example areas of application include:

- Internet security [WPY19]. Here, Google describes how their Chrome web browser incorporates MPC to warn users of potentially weak passwords.
- Problems in finance [BTW12]. Here, researchers used MPC to aggregate and study financial data from a number of tech companies.
- Secure auctions [BCD+09]. Here, in the oldest application of MPC, researchers arranged a privacy-preserving auction for the Danish sugar beet industry. Here, the auction arranged that farmers could bid on sugar beet contracts in a privacy-preserving manner, allowing them to hide their individual productivity.
- Healthcare [DRW+21]. Here, researchers demonstrated the potential to use MPC to collect and study sensitive patient data from various healthcare providers.
- Law enforcement [TMSgD22]. Here, researchers demonstrated a system that allows different law enforcement agencies to aggregate and analyze sensitive data from law enforcement databases.
- Privacy-preserving studies [BKK+15]. Here, researchers used MPC to study connections between tax and education records in Estonia.
- And many others.

More generally, because of its highly generic nature, MPC can provide privacy in the context of essentially any scenario where more than one entity wishes to join and analyze their data.

What We Will Not Study

Classically, cryptography considers the problem of secure communication. We consider two parties—usually called Alice and Bob—who wish to communicate in the presence of an untrusted adversary—usually called Eve. This setting is not the one we will consider in this course. Indeed, we will, unless otherwise stated, assume that the problem of secure communication has been solved. Our parties have secure point-to-point communication channels and can freely communicate.

What We Will Study

In this course, we will instead consider problems of the type where Alice (resp. Bob) is herself (resp. himself) the adversary. Namely, we wish to protect Bob (resp. Alice) from Alice who wishes to learn as much as possible about Bob's (resp. Alice's) input.

We will start from foundational feasibility results and move towards practical considerations of efficiency, in a variety of settings. We will focus on asymptotic results for general-purpose MPC, and we will also use MPC as a mechanism by which to study results in other related areas of cryptography.

The Cost of MPC

In fact, the cryptographic community has known general-purpose MPC protocols since the 1980s [Yao86, GMW87]. So why has MPC not seen broader adoption? What's the catch?

The problem with MPC is primarily one of inefficiency. As we will learn, MPC requires that the parties execute their desired program by means of a cryptographic protocol, and this can require converting each of the very lowest level operations of the program (think computing the AND or OR of two bits) to some kind of cryptographic operation. This can mean that an MPC protocol might run thousands of times slower—or worse—than if we were to run the same program on cleartext data.

Indeed, the earliest results in the field—which we will study first—were mere feasibility results: It is *technically possible* to securely run any program amongst any number of parties where the runtime of each party is polynomial in (1) the number of parties and (2) the size of program description. Feasibility results often do not translate to truly practical algorithms and protocols, and this is indeed the case in MPC. Modern research on MPC and related fields is focused on improving the efficiency and applicability of our protocols, with an end goal of achieving reasonably-efficient and easily-deployed cryptographic protocols for arbitrary programs.

Cryptographic cost comes in three main kinds:

- **Bandwidth consumption.** How many bits of information must the parties exchange to complete the cryptographic protocol?
- **Round complexity.** How many rounds of interaction are required before the cryptographic protocol completes? Note that round complexity is distinct from bandwidth consumption. Parties might exchange huge numbers of bits in one round. Round complexity measures the number of times parties must stop to wait for responses from their peers.
- **Computation.** What is the runtime of each party?

In this class we will examine ways to optimize these metrics.

In addition to the above quantitative metrics, one might look at other qualitative characteristics of MPC protocols:

- How many parties are supported? In particular, we will see that some protocols naturally support only two parties, while others scale to any number of parties. This difference can dramatically affect protocol performance.
- Which cryptographic primitives are required? Often in cryptography, we are unable to achieve provable security without first making *some* assumption about the complexity of computation. It is desirable to make such assumptions as mild as possible.
- What attacks are possible? In this course, we will think of some of the parties as being controlled by an adversary, and we will study various "kinds" of adversary. Some questions here include: How many parties can the adversary control? Does the adversary actively try to disrupt the cryptographic protocol? Is the adversary computationally bounded? Different answers to these questions can give protocols with different levels of efficiency.
- Which programs are supported? While general-purpose MPC protocols are known, other protocols target a specific type of computation. By creating a specialized protocol, one can achieve significantly better performance. For instance, the community has generated extremely efficient protocols for private set intersection, see e.g. [MAL23].

A Toy Protocol: Secure Sum

Let's start the course informally by considering a simple protocol that demonstrates "some kind" of secure computation is possible. Let us say that there are p parties P_i, each of whom has a number $x_i \in \{0, \ldots, 99\}$. We wish to securely compute and learn the sum $\sum_i x_i$.

We will consider a very weak kind of attack where no two parties work together, and where each party will follow the protocol precisely as we describe it. This is of course unrealistic, and we would like to consider a much stronger adversary, but this simple setting will serve as a useful starting point.

Here is a simple protocol that "works":

- First, the party P_0 chooses a very large random number `mask`, specifically a number that is far, far larger than the party inputs/computed output. For instance, let us say that `mask` is chosen uniformly at random between 0 and 2^{60}.
- Next, P_0 computes $y_0 = \texttt{mask} + x_0$ and sends y_0 to P_1. Notice that from P_1's perspective, y_0 "looks random". Indeed, `mask` hides P_0's input x_0, and so P_1 learns essentially nothing about x_0.
- Next P_1 computes $y_1 = y_0 + x_1$ and sends y_1 to P_2.
- Parties proceed in this fashion, with each P_i adding x_i to the sum y_{i-1} then sending it to P_{i+1} until...
- The last party P_{p-1} computes the sum y_{p-1}. Notice that $y_{p-1} = \texttt{mask} + \sum_i x_i$. P_{p-1} sends y_{p-1} to P_0.
- Remember that P_0 knows `mask`, so P_0 can subtract `mask` and learn $\sum_i x_i$.
- P_0 now sends $\sum_i x_i$ to all parties, and they each output the sum.

This completes the protocol. A few remarks:

- The protocol is clearly correct. The parties indeed learn $\sum_i x_i$.
- The protocol provides some kind of security, since the partial sum observed by party P_i is hidden by the large random number mask.
- We can measure the efficiency of the protocol. It runs in $O(p)$ rounds, uses $O(p)$ bits of communication, and each party performs a constant number of additions/subtractions.
- The protocol does not hide the fact that the parties computed a sum. It is a common misconception that the parties should not learn that a sum is being computed. However, recall that our setup here is that the parties *agreed* to compute some function f (in this case, the sum), and so this need not be hidden.
- The protocol is brittle. As one example, suppose two parties P_{i-1} and P_{i+1} collude. These parties can learn the input x_i of party P_i. **Can you see how?**

The following exercises are simply for practice:

Exercise 1. *Show how two colluding parties P_{i-1} and P_{i+1} can obtain party P_i's input x_i.*

Exercise 2. *Suppose the number of parties p is two. What does each party learn? Is the protocol secure when $p = 2$?*

Exercise 3. *A protocol that runs in $O(p)$ rounds is quite slow. Design a protocol for the same problem and the same setting that uses $O(p)$ bits of communication, but only $O(\log p)$ communication rounds. Argue why each party P_i learns nothing beyond their own input and the output.*

Here's another example secure protocol that we will discuss at some length next time. The following is a two-party protocol that computes the exclusive or (XOR) of two parties input bits. The following protocol is secure, assuming the parties follow the procedure:

Protocol 1 (Two-Party XOR)**.** *The protocol runs between two parties A and B.*

- ***Input:*** *Party A inputs a bit $x \in \{0,1\}$; B inputs a bit $y \in \{0,1\}$.*
- ***Output:*** *Both parties output $x \oplus y$.*

- ***Procedure:***
  - *A sends x to B.*
  - *B sends y to A.*
  - *A (resp. B) locally computes $x \oplus y$ and outputs the result.*

It is perhaps surprising that this protocol is secure – the parties simply send their inputs in the clear! However, upon further inspection it becomes intuitive why security holds: given her input x and the output $x \oplus y$, A can *compute* y. Thus, the fact that the parties agreed to compute XOR means that A is implicitly allowed to learn y (and similarly for B learning x). We will discuss this in some depth next time.

Next Time

We will begin to introduce our formal notions of security, with an initial goal of fully formalizing and characterizing the above protocol. Then, we will soon move on to more powerful cryptographic techniques for executing more interesting computations.

References

[BCD+09] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In Roger Dingledine and Philippe Golle, editors, *FC 2009*, volume 5628 of *LNCS*, pages 325–343. Springer, Berlin, Heidelberg, February 2009.

[BKK+15] Dan Bogdanov, Liina Kamm, Baldur Kubo, Reimo Rebane, Ville Sokk, and Riivo Talviste. Students and taxes: a privacy-preserving social study using secure computation. Cryptology ePrint Archive, Report 2015/1159, 2015.

[BTW12] Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis - (short paper). In Angelos D. Keromytis, editor, *FC 2012*, volume 7397 of *LNCS*, pages 57–64. Springer, Berlin, Heidelberg, February / March 2012.

[DRW+21] Xiao Dong, David A. Randolph, Chenkai Weng, Abel N. Kho, Jennie M. Rogers, and Xiao Wang. Developing high performance secure multi-party computation protocols in healthcare. In *AMIA Summits on Translational Science Proceedings*, 2021.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[MAL23] Daniel Morales, Isaac Agudo, and Javier Lopez. Private set intersection: A systematic literature review. *Computer Science Review*, 49:100570, 2023.

[TMSgD22] Amos Treiber, Dirk Müllmann, Thomas Schneider, and Indra Spiecker genannt Döhmann. Data protection law and multi-party computation: Applications to information exchange between law enforcement agencies. Cryptology ePrint Archive, Report 2022/1242, 2022.

[WPY19] Amanda Walker, Sarvar Patel, and Moti Yung. Helping organizations do more without collecting more data. Google Online Security Blog, 2019. `https://security.googleblog.com/2019/06/helping-organizations-do-more-without-collecting-more-data.html`.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.