Secure Multiparty Computation (MPC)

Yehuda Lindell

Unbound Tech and Bar-Ilan University
yehuda.lindell@unboundtech.com,lindell@biu.ac.il

Abstract. Protocols for secure multiparty computation (MPC) enable a set of parties to interact and compute a joint function of their private inputs while revealing nothing but the output. The potential applications for MPC are huge: privacy-preserving auctions, private DNA comparisons, private machine learning, threshold cryptography, and more. Due to this, MPC has been an intensive topic of research in academia ever since it was introduced in the 1980s by Yao for the two-party case (FOCS 1986), and by Goldreich, Micali and Wigderson for the multiparty case (STOC 1987). Recently, MPC has become efficient enough to be used in practice, and has made the transition from an object of theoretical study to a technology being used in industry. In this article, we will review what MPC is, what problems it solves, and how it is being currently used.
We note that the examples and references brought in this review article are far from comprehensive, and due to the lack of space many highly relevant works are not cited.

1 Introduction

Distributed computing considers the scenario where a number of distinct, yet connected, computing devices (or parties) wish to carry out a joint computation of some function. For example, these devices may be servers who hold a distributed database system, and the function to be computed may be a database update of some kind. The aim of *secure multiparty computation* is to enable parties to carry out such distributed computing tasks in a secure manner. Whereas distributed computing often deals with questions of computing under the threat of machine crashes and other inadvertent faults, secure multiparty computation is concerned with the possibility of deliberately malicious behaviour by some adversarial entity (these have also been considered in the distributed literature where they are called Byzantine faults). That is, it is assumed that a protocol execution may come under "attack" by an external entity, or even by a subset of the participating parties. The aim of this attack may be to learn private information or cause the result of the computation to be incorrect. Thus, two important requirements on any secure computation protocol are *privacy* and *correctness*. The privacy requirement states that nothing should be learned beyond what is absolutely necessary; more exactly, parties should learn their output and nothing else. The correctness requirement states that each party should receive its correct output. Therefore, the adversary must not be able to cause the result of the computation to deviate from the function that the parties had set out to compute.

Secure multiparty computation can be used to solve a wide variety of problems, enabling the utilisation of data without compromising privacy. Consider, for example, the problem of comparing a person's DNA against a database of cancer patients' DNA, with the goal of finding if the person is in a high risk group for a certain type of cancer. Such a task clearly has important health and societal benefits. However, DNA information is highly sensitive, and should not be revealed to private organisations. This dilemma can be solved by running a secure multiparty computation that

reveals only the category of cancer that the person's DNA is close to (or none). In this example, the privacy requirement ensures that only the category of cancer is revealed, and nothing else about anyone's DNA (neither the DNA of the person being compared nor the DNA of the patients in the database). Furthermore, the correctness requirement guarantees that a malicious party cannot change the result (e.g., make the person think that they are at risk of a type of cancer, and therefore need screening).

In another example, consider a trading platform where parties provide offers and bids, and are matched whenever an offer is greater than a bid (with, for example, the price of the trade being some function of the offer and bid prices). In such a scenario, it can be beneficial from a game theoretic perspective to not reveal the parties' actual offers and bids (since this information can be used by others in order to artificially raise prices or provide bids that are lower than their utility). Privacy here guarantees that only the match between buyer and seller and the resulting price is revealed, and correctness would guarantee that the price revealed is the correct one according to the function (and not some lower value, for example). It is interesting to note that in some cases privacy is more important (like in the DNA example), whereas in others correctness is more important (like in the trading example). In any case, MPC guarantees both of these properties, and more.

A note on terminology. In the literature, beyond secure multiparty computation (with acronym MPC, and sometimes SMPC), there are also references to secure function evaluation (SFE). These notions overlap significantly, and are often used synonymously. In addition, special cases of MPC often have their own names. Two examples are private set intersection (PSI) which considers the secure computation of the intersection of private sets, and threshold cryptography which considers the secure computation of digital signatures and decryption, where no single party holds the private key.

2 Security of MPC

2.1 The Definitional Paradigm

As we have mentioned above, the setting that we consider is one where an adversarial entity controls some subset of the parties and wishes to attack the protocol execution. The parties under the control of the adversary are called corrupted, and follow the adversary's instructions. Secure protocols should withstand any adversarial attack (where the exact power of the adversary will be discussed later). In order to formally claim and prove that a protocol is secure, a precise definition of security for multiparty computation is required. A number of different definitions have been proposed and these definitions aim to ensure a number of important security properties that are general enough to capture most (if not all) multiparty computation tasks. We now describe the most central of these properties:

1. *Privacy:* No party should learn anything more than its prescribed output. In particular, the only information that should be learned about other parties' inputs is what can be derived from the output itself. For example, in an auction where the only bid revealed is that of the highest bidder, it is clearly possible to derive that all other bids were lower than the winning bid. However, nothing else should be revealed about the losing bids.
2. *Correctness:* Each party is guaranteed that the output that it receives is correct. To continue with the example of an auction, this implies that the party with the highest bid is guaranteed to win, and no party including the auctioneer can influence this.

3. *Independence of Inputs:* Corrupted parties must choose their inputs independently of the honest parties' inputs. This property is crucial in a sealed auction, where bids are kept secret and parties must fix their bids independently of others. We note that independence of inputs is *not* implied by privacy. For example, it may be possible to generate a higher bid, without knowing the value of the original one. Such an attack can actually be carried out on some encryption schemes (i.e., given an encryption of \$100, it is possible to generate a valid encryption of \$101, without knowing the original encrypted value).
4. *Guaranteed Output Delivery:* Corrupted parties should not be able to prevent honest parties from receiving their output. In other words, the adversary should not be able to disrupt the computation by carrying out a "denial of service" attack.
5. *Fairness:* Corrupted parties should receive their outputs if and only if the honest parties also receive their outputs. The scenario where a corrupted party obtains output and an honest party does not should not be allowed to occur. This property can be crucial, for example, in the case of contract signing. Specifically, it would be very problematic if the corrupted party received the signed contract and the honest party did not. Note that guaranteed output delivery implies fairness, but the converse is not necessarily true.

We stress that the above list does *not* constitute a definition of security, but rather a set of requirements that should hold for any secure protocol. Indeed, one possible approach to defining security is to just generate a list of separate requirements (as above) and then say that a protocol is secure if all of these requirements are fulfilled. However, this approach is not satisfactory for the following reasons. First, it may be possible that an important requirement was missed. This is especially true because different applications have different requirements, and we would like a definition that is general enough to capture all applications. Second, the definition should be simple enough so that it is trivial to see that *all* possible adversarial attacks are prevented by the proposed definition.

The standard definition today [5] therefore formalizes security in the following general way. As a mental experiment, consider an "ideal world" in which an external trusted (and incorruptible) party is willing to help the parties carry out their computation. In such a world, the parties can simply send their inputs to the trusted party, who then computes the desired function and passes each party its prescribed output. Since the only action carried out by a party is that of sending its input to the trusted party, the only freedom given to the adversary is in choosing the corrupted parties' inputs. Notice that all of the above-described security properties (and more) hold in this ideal computation. For example, privacy holds because the only message ever received by a party is its output (and so it cannot learn any more than this). Likewise, correctness holds since the trusted party cannot be corrupted and so will always compute the function correctly.

Of course, in the "real world", there is no external party that can be trusted by all parties. Rather, the parties run some protocol amongst themselves without any help, and some of them are corrupted and colluding. Despite this, a secure protocol should emulate the so-called "ideal world". That is, a real protocol that is run by the parties (in a world where no trusted party exists) is said to be secure, if no adversary can do more harm in a real execution that in an execution that takes place in the ideal world. This can be formulated by saying that for any adversary carrying out a successful attack in the real world, there exists an adversary that successfully carries out an attack with the same effect in the ideal world. However, successful adversarial attacks *cannot* be carried out in the ideal world. We therefore conclude that all adversarial attacks on protocol executions in the real world must also fail.

More formally, the security of a protocol is established by comparing the outcome of a real protocol execution to the outcome of an ideal computation. That is, for any adversary attacking

a real protocol execution, there exists an adversary attacking an ideal execution (with a trusted party) such that the input/output distributions of the adversary and the participating parties in the real and ideal executions are essentially the same. Thus a real protocol execution "emulates" the ideal world. This formulation of security is called the ideal/real simulation paradigm. In order to motivate the usefulness of this definition, we describe why all the properties described above are implied. Privacy follows from the fact that the adversary's output is the same in the real and ideal executions. Since the adversary learns nothing beyond the corrupted party's outputs in an ideal execution, the same must be true for a real execution. Correctness follows from the fact that the honest parties' outputs are the same in the real and ideal executions, and from the fact that in an ideal execution, the honest parties all receive correct outputs as computed by the trusted party. Regarding independence of inputs, notice that in an ideal execution, all inputs are sent to the trusted party before any output is received. Therefore, the corrupted parties know nothing of the honest parties' inputs at the time that they send their inputs. In other words, the corrupted parties' inputs are chosen independently of the honest parties' inputs, as required. Finally, guaranteed output delivery and fairness hold in the ideal world because the trusted party always returns all outputs. The fact that it also holds in the real world again follows from the fact that the honest parties' outputs are the same in the real and ideal executions.

We remark that in some cases, the definition is relaxed to exclude fairness and guaranteed output delivery. The level of security achieved when these are excluded is called "security with abort", and the result is that the adversary may be able to obtain output while the honest parties do not. There are two main reasons why this relaxation is used. First, in some cases, it is impossible to achieve fairness (e.g., it is impossible to achieve fair coin tossing for two parties [11]). Second, in some cases, more efficient protocols are known when fairness is not guaranteed. Thus, if the application does not require fairness (and in particular in cases where only one party receives output), this relaxation is helpful.

2.2 Additional Definitional Parameters

Adversarial power. The above informal definition of security omits one very important issue: the power of the adversary that attacks a protocol execution. As we have mentioned, the adversary controls a subset of the participating parties in the protocol. However, we have not defined what power such an adversary has. We describe the two main parameters defining the adversary: its allowed adversarial behaviour (i.e., does the adversary just passively gather information or can it instruct the corrupted parties to act maliciously) and its corruption strategy (i.e., when or how parties come under the "control" of the adversary):

1. **Allowed adversarial behaviour:** The most important parameter that must be defined relates to the actions that corrupted parties are allowed to take. There are three main types of adversaries:
   (a) Semi-honest adversaries: In the semi-honest adversarial model, even corrupted parties correctly follow the protocol specification. However, the adversary obtains the internal state of all the corrupted parties (including the transcript of all the messages received), and attempts to use this to learn information that should remain private. This is a rather weak adversarial model, but a protocol with this level of security does guarantee that there is no inadvertent data leakage. In some cases this is sufficient, although in today's adversarial environment it

is often insufficient. Semi-honest adversaries are also called "honest-but-curious" and "passive". (Sometimes, fail-stop adversaries are also considered; these are essentially semi-honest adversaries who may also halt the protocol execution early.)

(b) Malicious adversaries: In this adversarial model, the corrupted parties can *arbitrarily* deviate from the protocol specification, according to the adversary's instructions. In general, providing security in the presence of malicious adversaries is preferred, as it ensures that no adversarial attack can succeed. Malicious adversaries are also called "active".

(c) Covert adversaries [1]: This type of adversary may behave maliciously in an attempt to break the protocol. However, the security guarantee provided is that if it does attempt such an attack, then it will be detected with some specified probability that can be tuned to the application. We stress that unlike in the malicious model, if the adversary is not detected then it may successfully cheat (e.g., learn an honest party's input). This model is suited to settings where some real-world penalty can be associated with an adversary being detected, and the adversary's expectation is to lose overall if it attempts an attack.

2. **Corruption strategy:** The corruption strategy deals with the question of when and how parties are corrupted. There are three main models:

(a) Static corruption model: In this model, the set of parties controlled by the adversary is fixed before the protocol begins. Honest parties remain honest throughout and corrupted parties remain corrupted.

(b) Adaptive corruption model: Rather than having a fixed set of corrupted parties, adaptive adversaries are given the capability of corrupting parties during the computation. The choice of who to corrupt, and when, can be arbitrarily decided by the adversary and may depend on its view of the execution (for this reason it is called adaptive). This strategy models the threat of an external "hacker" breaking into a machine during an execution, or a party who is honest initially and later changes its behaviour. We note that in this model, once a party is corrupted, it remains corrupted from that point on.

(c) Proactive security model [30,7]: This model considers the possibility that parties are corrupted for a certain period of time only. Thus, honest parties may become corrupted throughout the computation (like in the adaptive adversarial model), but corrupted parties may also become honest. The proactive model makes sense in cases where the threat is an external adversary who may breach networks and break into services and devices, and secure computations are ongoing. When breaches are discovered, the systems are cleaned and the adversary loses control of some of the machines, making the parties honest again. The security guarantee is that the adversary can only learn what it derived from the local state of the machines that it corrupted, while they were corrupted. Such an adversary is sometimes called mobile.

There is no "right" model when considering the above. Rather, the specific definition used and adversary considered depends on the application and the threats being dealt with.

Modular sequential and concurrent composition. In reality, a secure multiparty computation protocol is not run in isolation; rather it is part of a system. In [5], it was proven that if you run an MPC protocol as part of a larger system, then it still behaves in the same way as if an incorruptible trusted party carried out the computation for the parties. This powerful theorem is called modular composition, and it enables larger protocols to be constructed in a modular way using secure subprotocols, as well as analysing a larger system that uses MPC for some of the computations.

One important question in this context is whether or not the MPC protocol itself runs at the same time as other protocols. In the setting of *sequential composition*, the MPC protocol can run

as a subprotocol of another protocol with arbitrary other messages being sent before and after the MPC protocol. However, the MPC protocol itself must be run without any other messages being sent in parallel. This is called the stand-alone setting, and is the setting considered by the basic definition of security of [5]. The sequential modular composition theorem of [5] states that in this setting, the MPC protocol indeed behaves like a computation carried out by a trusted third party.

In some (many) cases, MPC protocols are run at the same time as other instances of itself, other MPC protocols, and other insecure protocols. In these cases, a protocol proven secure under the aforementioned stand-alone definition of security may not actually remain secure. A number of definitions were proposed to deal with this setting, the most popular of these is that of universal composability [6]. Any protocol proven secure according to this definition is guaranteed to behave like an ideal execution, irrespective of what other protocols run concurrently to it. As such, this is the gold standard of MPC definitions. However, it does come at a price (both of efficiency, and of assumptions required on the system setup).

2.3 Important Definitional Implications

The ideal model and using MPC in practice. The ideal/real paradigm for defining security actually has some very important implications for the use of MPC in practice. Specifically, in order to *use* an MPC protocol, all a practitioner needs to do is to consider the security of their system when an incorruptible trusted party carries out the computation for which MPC is used. If the system is secure in this case, then it will remain secure even when the real MPC protocols is used (under the appropriate composition case). This means that non-cryptographers need not understand anything about *how* MPC protocols work, or even how security is defined. The ideal model provides a clean and easy to understand abstraction that can be utilised by those constructing systems.

Any inputs are allowed. Although the ideal model paradigm provides a simple abstraction, as described above, there is a subtle point that is sometime misunderstood. An MPC protocol behaves like an ideal execution; as such, the security obtained is analogous to that of an ideal execution. However, in an ideal execution, adversarial parties may input any values that they wish, and indeed there is no generic way of preventing this. Thus, if two people wish to see who earns a higher salary (without revealing any more than this one bit of information), then nothing stops one of them from inputting the maximum possible value as their salary (and then behaving honestly in the MPC protocol itself), with the result being that they earn more. Thus, if the security of an application depends on the party's using *correct inputs*, then mechanisms must be used to enforce this. For example, it is possible to require signed inputs, and have the signature be verified as part of the MPC computation. Depending on the specific protocol, this can add significant cost.

MPC secures the process, but not the output. Another subtlety that is often misunderstood is that MPC secures the process, meaning that nothing is revealed by the computation itself. However, this does not mean that the output of the function being computed does not reveal sensitive information. For an extreme example, consider two people computing the average of their salaries. It is indeed true that nothing but the average will be output, but given a person's own salary and the average of both salaries, they can derive the exact salary of the other person. Thus, just using MPC does not mean that all privacy concerns are solved. Rather, MPC secures the computing process, and the question of what functions should and should not be computed due to privacy concerns still needs to be addressed. In some cases, like threshold cryptography, this question is not an issue (since the output of cryptographic functions does not reveal the key, assuming that it's secure). However, in other cases, it may be less clear.

3 Feasibility of MPC

The above-described definition of security seems to be very restrictive in that no adversarial success is tolerated, and the protocol should behave as if a trusted third party is carrying out the computation. Thus, one may wonder whether it is even possible to obtain secure protocols under this definition, and if yes, for which distributed computing tasks. Perhaps surprisingly, powerful feasibility results have been established, demonstrating that in fact, *any* distributed computing task (function) can be securely computed, in the presence of malicious adversaries. We now briefly state the most central of these results. Let n denote the number of participating parties and let t denote a bound on the number of parties that may be corrupted (where the identity of the corrupted parties is unknown):

1. For $t < n/3$ (i.e., when less than a third of the parties can be corrupted), secure multiparty protocols with fairness and guaranteed output delivery can be achieved for any function with computational security assuming a synchronous point-to-point network with authenticated channels [18], and with information-theoretic security assuming the channels are also private [3,9].
2. For $t < n/2$ (i.e., in the case of a guaranteed honest majority), secure multiparty protocols with fairness and guaranteed output delivery can be achieved for any function with computational and information-theoretic security, assuming that the parties also have access to a broadcast channel [18,33].
3. For $t \geq n/2$ (i.e., when the number of corrupted parties is not limited), secure multiparty protocols (without fairness or guaranteed output delivery) can be achieved [37,18].

In the setting of concurrent composition described at the end of Section 2.2, it has also been shown that any function can be securely computed [6,8].

In summary, secure multiparty protocols exist for any distributed computing task. This fact is what provides its huge potential – whatever needs to be computed can be computed securely! We stress, however, that the aforementioned feasibility results are *theoretical*, meaning that they demonstrate that this is possible in principle. They do not consider the practical efficiency costs incurred; these will be mentioned below in Section 4.6.

We conclude this section with a caveat. The above feasibility results are proven in specific models, and under cryptographic hardness and/or setting assumptions. It is beyond the scope of this review to describe these details, but it is important to be aware that they need to be considered.

4 Techniques

Over the past three decades, many different techniques have been developed for constructing MPC protocols with different properties, and for different settings. It is way beyond the scope of this paper to even mention all of the techniques, and we highly recommend reading [15] for an extremely well-written and friendly introduction to MPC, including a survey of the major techniques. Nevertheless, we will provide a few simple examples of how MPC protocols are constructed, in order to illustrate how it can work.

4.1 Shamir Secret Sharing

MPC protocols for an honest majority typically utilise secret sharing as a basic tool. We will therefore begin by briefly describing Shamir's secret sharing scheme [34].

A secret sharing scheme solves the problem of a dealer who wishes to share a secret s amongst n parties, so that any subset of $t+1$ or more of the parties can reconstruct the secret, yet no subset of t or fewer parties can learn anything about the secret. A scheme that fulfils these requirements is called a $(t+1)$-out-of-n-threshold secret-sharing scheme.

Shamir's secret sharing scheme utilises the fact that for any for $t+1$ points on the two dimensional plane $(x_1, y_1), \ldots, (x_{t+1}, y_{t+1})$ with unique x_i, there exists a unique polynomial $q(x)$ of degree at most t such that $q(x_i) = y_i$ for every i. Furthermore, it is possible to efficiently reconstruct the polynomial $q(x)$, or any specific point on it. One way to do this is with the Lagrange basis polynomials $\ell_1(x), \ldots, \ell_t(x)$, where reconstruction is carried out by computing $q(x) = \sum_{i=1}^{t+1} \ell_i(x) \cdot y_i$. From here on, we will assume that all computations are in the finite field $\mathbb{Z}_p$, for a prime $p > n$.

Given the above, in order to share a secret s, the dealer chooses a random polynomial $q(x)$ of degree at most t under the constraint that $q(0) = s$. (Concretely, the dealer sets $a_0 = s$ and chooses random coefficients $a_1, \ldots, a_t \in \mathbb{Z}_p$, and sets $q(x) = \sum_{i=0}^{t} a_i \cdot x^i$.) Then, for every $i = 1, \ldots, n$, the dealer provides the i'th party with the share $y_i = q(i)$; this is the reason why we need $p > n$, so that different shares can be given to each party. Reconstruction by a subset of any t parties works by simply interpolating the polynomial to compute $q(x)$ and then deriving $s = q(0)$. Although $t+1$ parties can completely recover s, it is not hard to show that *any* subset of t or fewer parties cannot learn anything about s. This is due to the fact that they have t or fewer points on the polynomial, and so there exists a polynomial going through these points and the point $(0, s)$ for every possible $s \in \mathbb{Z}_p$. Furthermore, since the polynomial is random, all polynomials are equally likely, and so all values of $s \in \mathbb{Z}_p$ are equally likely.

4.2 Honest-Majority MPC with Secret Sharing

The first step in most protocols for *general* MPC (i.e., protocols that can be used to compute any function) is to represent the function being computed as a Boolean or arithmetic circuit. In the case of honest-majority MPC based on secret sharing, the arithmetic circuit (comprised of multiplication and addition gates) is over a finite field $\mathbb{Z}_p$ with $p > n$, as above. We remark that arithmetic circuits are Turing complete, and so any function can be represented in this form. The parties participating in the MPC protocol are all provided this circuit, and we assume that they can all communicate securely with each other. The protocol for semi-honest adversaries (see below for what is needed for the case of malicious adversaries) consists of the following phases:

1. **Input sharing:** In this phase, each party shares its input with the other parties, using Shamir's secret sharing. That is, for each input wire to the circuit, the party whose input is associated with that wire plays the dealer in Shamir's secret sharing to share the value to all parties. The secret sharing used is $(t+1)$-out-of-n, with $t = \left\lfloor \frac{n-1}{2} \right\rfloor$ (thus, the degree of the polynomial is t). This provides security against any minority of corrupted parties, since no such minority can learn anything about the shared values. Following this step, the parties hold secret shares of the values on each input wire.
2. **Circuit evaluation:** In this phase, the parties evaluate the circuit one gate at a time, from the input gates to the output gates. The evaluation maintains the invariant that for every gate for which the parties hold $(t+1)$-out-of-n sharings of the values on the two input wires, the result of the computation is a $(t+1)$-out-of-n secret sharing of the value on the output wire of the gate.
   (a) *Computing addition gates:* According to the invariant, each party holds a secret sharing of the values on the input wires to the gate; we denote these polynomials by $a(x)$ and $b(x)$ and this means that the i'th party holds the values $a(i)$ and $b(i)$. The output wire of this gate

should be a $(t+1)$-out-of-n secret sharing of the value $a(0)+b(0)$. This is easily computed by the i'th party locally setting its share on the output wire to be $a(i)+b(i)$. Observe that by defining the polynomial $c(x) = a(x)+b(x)$, this means that the i'th party holds $c(i)$. Furthermore, $c(x)$ is a degree-t polynomial such that $c(0) = a(0)+b(0)$. Thus, the parties hold a valid $(t+1)$-out-of-n secret sharing of the value $a(0)+b(0)$, as required. Observe that no communication is needed in order to compute addition gates.

(b) *Computing multiplication gates:* Once again, denote the polynomials on the input wires to the gate by $a(x)$ and $b(x)$. As for an addition gate, the i'th party can locally multiply its shares to define $c(i) = a(i) \cdot b(i)$. By the properties of polynomial multiplication, this defines a polynomial $c(x)$ such that $c(0) = a(0) \cdot b(0)$. Thus, $c(x)$ is a sharing of the correct value (the product of the values on the input wires). However, $c(x)$ is of degree-$2t$, and thus this is a $(2t+1)$-out-of-n secret sharing and not a $(t+1)$-out-of-n secret sharing. In order to complete the computation of the multiplication gate, it is therefore necessary for the parties to carry out a *degree reduction* step, to securely reduce the degree of the polynomial shared amongst the parties from $2t$ to t, without changing its value at 0. Before proceeding to describe this, observe that since $t < n/2$, the shares held by the n parties do fully determine the polynomial $c(x)$ of degree $2t+1$.

In order to compute the degree reduction step, we use an idea from [12] (we describe the basic idea here, although [12] have a far more efficient way of realising it than what we describe here). Assume that the parties all hold two independent secret sharings of an unknown random value r, the first sharing via a polynomial of degree-$2t$ denoted $R_{2t}(x)$, and the second sharing via a polynomial of degree-t denoted $R_t(x)$. Note that $R_{2t}(0) = R_t(0) = r$. Then, each party can locally compute its share of the degree-$2t$ polynomial $d(x) = c(x) - R_{2t}(x)$ by setting $d(i) = c(i) - R_{2t}(i)$. Note that both $c(x)$ and $R_{2t}(x)$ are of degree-$2t$. Next, the parties reconstruct $d(0) = a(0) \cdot b(0) - r$ by sending all of their shares to all other parties. Finally, the i'th party for all $i = 1, \ldots, n$ computes its share on the output wire to be $c'(i) = R_t(i) + d(0)$.

Observe that $c'(x)$ is of degree t since $R_t(x)$ is of degree t, and it is defined by adding a constant $d(0)$ to $R_t(x)$. Next, $c'(0) = a(0) \cdot b(0)$ since $R_t(0) = r$ and $d(0) = a(0) \cdot b(0) - r$; thus r cancels out when summing the values. Thus, the parties hold a valid $(t+1)$-out-of-n secret sharing of the product of the values on the input wires, as required. Furthermore, note that the value $d(0)$ that is revealed to all parties does not leak any information since $R_t(x)$ perfectly masks all values of $c(x)$, and in particular it masks the value $a(0) \cdot b(0)$.

It remains to show how the parties generate two independent secret sharings of an unknown random value r, via polynomials of degree $2t$ and t. This can be achieved by the i'th party, for all $i = 1, \ldots, n$, playing the dealer and sharing a random value r_i via a degree-$2t$ polynomial $R^i_{2t}(x)$ and via a degree-t polynomial $R^i_t(x)$. Then, upon receiving such shares from each of the parties, the i'th party for all $i = 1, \ldots, n$ defines its shares of $R_{2t}(x)$ and $R_t(x)$ by computing $R_{2t}(i) = \sum_{j=1}^{n} R^j_{2t}(i)$ and $R_t(i) = \sum_{j=1}^{n} R^j_t(i)$. Since all parties contribute secret random values $r_1, \ldots, r_n$ and we have that $r = \sum_{j=1}^{n} r_j$, it follows that no party knows r.

3. **Output reconstruction:** Once the parties have obtained shares on the output wires, they can obtain the outputs by simply sending their shares to each other and reconstructing the outputs via interpolation. Observe that it is also possible for different parties to obtain different outputs, if desired. In this case, the parties send the shares for reconstruction only to the relevant parties who are supposed to obtain the output on a given wire.

The above protocol is secure for *semi-honest adversaries*, as long as less than $n/2$ parties are corrupted. This is because the only values seen by the parties during the computation are secret shares (that reveal nothing about the values they hide), and opened $d(0)$ values that reveal nothing about the actual values on the wires due to the independent random sharings used each time. Note that in order to achieve security in the presence of *malicious adversaries* who may deviate from the protocol specification, it is necessary to utilise different methods to prevent cheating. See [4,10,16] for a few examples of how to efficiently achieve security in the presence of malicious adversaries.

4.3 Private Set Intersection

In Section 4.2, we described an approach to *general* secure computation that can be used to securely compute any function. In many cases, these general approaches turn out to actually be the most efficient (especially when considering malicious adversaries). However, in some cases, the specific structure of the function being solved enables us to find faster, tailored solutions. In this and the next section, we present two examples of such functions.

In a private set intersection protocol, two parties with private sets of values wish to find the intersection of the sets, without revealing anything but the elements in the intersection. In some cases, some function of the intersection is desired, like its size only. There has been a lot of work on this problem, with security for both semi-honest and malicious adversaries, and with different efficiency goals (few rounds, low communication, low computation, etc.). In this section, we will describe the idea behind the protocol of [23]; the actual protocol of [23] is far more complex, but we present the conceptually simple idea underlying their construction.

A pseudorandom function F is a keyed function with the property that outputs of the function on known inputs look completely random. Thus, for any given list of elements $x_1, \ldots, x_n$, the series of values $F_k(x_1), \ldots, F_k(x_n)$ looks random. In particular, given $F_k(x_i)$, it is infeasible to determine the value of x_i. In the following simple protocol, we utilise a tool called *oblivious pseudorandom function evaluation*. This is a specific type of MPC protocol where the first party inputs k and the second party inputs x, and the second party receives $F_k(x)$ while the first party learns nothing about x (note that the second party learns $F_k(x)$ but nothing beyond that; in particular, k remains secret). Such a primitive can be built in many ways, and we will not describe them here.

Now, consider two parties with respective sets of private elements; denote them $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$, respectively (for simplicity, we assume that their lists are of the same size, although this is not needed). Then, the protocol proceeds as follows:

1. The first party chooses a key k for a pseudorandom function.
2. The two parties run n oblivious pseudorandom function evaluations: in the ith execution, the first party inputs k and the second party inputs y_i. As a result, the second party learns $F_k(y_1), \ldots, F_k(y_n)$ while the first party learns nothing about $y_1, \ldots, y_n$.
3. The first party locally computes $F_k(x_1), \ldots, F_k(x_n)$ and sends the list to the second party. It can compute this since it knows k.
4. The second party computes the intersection between the lists $F_k(y_1), \ldots, F_k(y_n)$ and $F_k(x_1), \ldots, F_k(x_n)$, and outputs all values y_j for which $F_k(y_j)$ is in the intersection. (The party knows these values since it knows the association between y_j and $F_k(y_j)$.)

The above protocol reveals nothing but the intersection since the first party learns nothing about $y_1, \ldots, y_n$ from the oblivious pseudorandom function evaluations, and the second party learns nothing about values of x_j that are not in the intersection since the pseudorandom function hides the

preimage values. This is therefore secure in the semi-honest model. It is more challenging to achieve security in the malicious model. For example, a malicious adversary could use a different key for the first element and later elements, and then have the result that the value y_1 is in the output if and only if it was the first element of the second party's list.

The most efficient private set intersection protocols today use advanced hashing techniques, and can process millions of items in a few seconds [23,32,31].

4.4 Threshold Cryptography

The aim of threshold cryptography is to enable a set of parties to carry out cryptographic operations, without any single party holding the secret key. This can be used to ensure multiple signatories on a transaction, or alternatively to protect secret keys from being stolen by spreading key shares out on different devices (so that the attacker has to breach all devices in order to learn the key). We demonstrate a very simple protocol for two-party RSA, but warn that for more parties (and other schemes), it is much more complex.

RSA is a public-key scheme with public-key (e, N) and private-key (d, N). The basic RSA function is $y = x^e \bmod N$, and its inverse function is $x = y^d \bmod N$. RSA is used for encryption and signing, by padding the message and other techniques. Here, we relate to the *raw* RSA function, and show how the inverse can be computed securely amongst two parties, where neither party can compute the function itself. In order to achieve this, the system is set up with the first party holding (d_1, N) and the second party holding (d_2, N), where d_1 and d_2 are random under the constraint that $d_1 + d_2 = d$. (More formally, the order in the exponent is $\phi(N)$ – Euler's function – and therefore the values $d_1, d_2 \in \mathbb{Z}_{\phi(N)}$ are random under the constraint that $d_1 + d_2 = d \bmod \phi(N)$.) In order to securely compute $y^d \bmod N$, the first party computes $x_1 = y^{d_1} \bmod N$, the second party computes $x_2 = y^{d_2} \bmod N$, and these values are exchanged between them. Then, each party computes $x = x_1 \cdot x_2 \bmod N$, verifies that the output is correct by checking that $x^e = y \bmod N$, and if yes outputs x. Observe that this computation is correct since

$$x = y^{d_1} \cdot y^{d_2} \bmod N = y^{d_1 + d_2 \bmod \phi(N)} \bmod N = y^d \bmod N.$$

In addition, observe that given the output x and its share d_1 of the private exponent, the first party can compute $x_2 = x/y^{d_1} \bmod N$ (this is correct since $x_2 = y^{d_2} = y^{d_1 + d_2 - d_1} = y^d \cdot y^{-d_1} = x/y^{d_1} \bmod N$). This means that the first party does not learn anything more than the output from the protocol, since it can generate the messages that it receives in the protocol by itself from its own input and the output.

We stress that full-blown threshold cryptography supports quorum approvals involving many parties (e.g., requiring $(t+1)$-out-of-n parties to sign, and maintaining security for any subset of t corrupted parties). This needs additional tools, but can also be done very efficiently; see, [35] and references within. Recently, there has been a lot of interest in threshold ECDSA, due to its applications to protecting cryptocurrencies [26,17,27,14].

4.5 Dishonest-Majority MPC

In Section 4.2, we described a general protocol for MPC that is secure as long as an adversary cannot corrupt more than a minority of the parties. In the case of a dishonest majority, including the important special case of two parties (with one corrupted), completely different approaches

are needed. There has been a very large body of work in this direction, from the initial protocols of [37,18,2] that focused on feasibility, and including a lot of recent work focused on achieving concrete efficiency. There is so much work in this direction that any attempt to describe it here will do it a grave injustice. We therefore refer the reader to [15] for a description of the main approaches, including the GMW oblivious transfer approach [18,21], garbled circuits [37,2], cut-and-choose [28], SPDZ [13], TinyOT [29], MPC in the head [22], and more. (We stress that for each of these approaches, there have been many follow-up works, achieving increasingly better efficiency.)

4.6 Efficient and Practical MPC

The first twenty years of MPC research focused primarily on feasibility: how to define and prove security for multiple adversarial and network models, under what cryptographic and setup assumptions it is possible to achieve MPC, and more. The following decade saw a large body of research around making MPC more and more efficient. The first steps of this process were purely algorithmic and focused on reducing the overhead of the cryptographic primitives. Following this, other issues were considered that had significant impact: the memory and communication, utilisation of hardware instructions like AES-NI, and more. In addition, since most general protocols require the circuit representation of the function being computed, and circuits are hard to manually construct, special-purpose MPC compilers from code to circuits were also constructed. These compilers are tailored to be sensitive to the special properties of MPC. For example, in many protocols XOR gates are computed almost for free [24], in contrast to AND/OR gates that cost. These compilers therefore minimise the number of AND gates, even at the expense of considerably more XOR gates. In addition, the computational cost of some protocols is dominated by the circuit size, while in others it is dominated by the circuit depth. Thus, some compilers aim to generate the smallest circuit possible, while others aim to generate a circuit with the lowest depth. See [19] for a survey on general-purpose compilers for MPC and their usability. The combination of these advancements led to performance improvements of many orders of magnitude in just a few years, paving the way for MPC to be fast enough to be used in practice for a wide variety of problems. See [15, Chapter 4] for a description of a few of the most significant of these advancements.

5 MPC Use Cases

There are many great theoretical examples of where MPC can be helpful. It can be used to compare no-fly lists in a privacy-preserving manner, to enable private DNA comparisons for medical and other purposes, to gather statistics without revealing anything but the aggregate results, and much more. Up until very recently, these theoretical examples of usage were almost all we had to say about the potential benefits of MPC. However, the situation today is very different. MPC is now being used in multiple real-world use cases, and usage is growing fast.

We will conclude this review paper with some examples of MPC applications that have been actually deployed.

Boston wage gap [25]. The Boston Women's Workforce Council used MPC in 2017 in order to compute statistics on the compensation of 166,705 employees across 114 companies, comprising roughly 16% of the Greater Boston area workforce. The use of MPC was crucial, since companies would not provide their raw data due to privacy concerns. The results showed that the gender gap in the Boston area is even larger than previously estimated by the U.S. Bureau of Labor Statistics. This is a powerful example demonstrating that MPC can be used for social good.

Advertising conversion [20]. In order to compute accurate conversion rates from advertisements to actual purchases, Google computes the *size* of the intersection between the list of people shown an advertisement to the list of people actually purchasing the advertised goods. When the goods are not purchased online and so the purchase connection to the shown advertisement cannot be tracked, Google and the company paying for the advertisement have to share their respective lists in order to compute the intersection size. In order to compute this without revealing anything but the size of the intersection, Google utilises a protocol for privacy-preserving set intersection. The protocol used by Google is described in [20]. Although this protocol is far from the most efficient known today, it is simple and meets their computational requirements.

MPC for cryptographic key protection [38]. As described in Section 4.4, threshold cryptography provides the ability to carry out cryptographic operations (like decryption and signing) without the private key being held in any single place. A number of companies are using threshold cryptography as an alternative to legacy hardware for protecting cryptographic keys. In this application, MPC is not run between different parties holding private information. Rather, a single organisation uses MPC to generate keys and compute cryptographic operations, without the key ever being in a single place where it can be stolen. By placing the key shares in different environments, it is very hard for an adversary to steal all shares and obtain the key. In this setting, the proactive model described in Section 2.2 is the most suitable. Another use of MPC in this context is for protecting the signing keys used for protecting cryptocurrencies and other digital assets. Here, the ability to define general quorums enables the cryptographic enforcement of strict policies for approving financial transactions, or to share keys between custody providers and clients.

Government collaboration [39]. Different governmental departments hold information about citizens, and significant benefit can be obtained by correlating that information. However, the privacy risks involved in pooling private information can prevent governments from doing this. For example, in 2000, Canada scrapped a program to pool citizen information, under criticism that they were building a "big brother database". Utilising MPC, Estonia collected encrypted income tax records and higher education records to analyse if students who work during their degree are more likely to fail than those focusing solely on their studies. By using MPC, the government was guaranteed that all data protection and tax secrecy regulations were followed without losing data utility.

Privacy-preserving analytics [40]. Machine learning usage is increasing rapidly in many domains. MPC can be used to run machine learning models on data without revealing the model (which contains precious intellectual property) to the data owner, and without revealing the data to the model owner. In addition, statistical analyses can be carried out between organisations for the purpose of anti money laundering, risk score calculations, and more.

6 Discussion

Secure multiparty computation is a fantastic example of success in the long game of research [36]. For the first 20 years of MPC research, no applications were in sight, and it was questionable whether or not MPC would ever be used. In the past decade, the state of MPC usability has undergone a radical transformation. In this time, MPC has not only become fast enough to be used in practice, but it has received industry recognition and has made the transition to a technology that is deployed in practice. MPC still requires great expertise to deploy, and additional research breakthroughs are

needed to make secure computation practical on large data sets and for complex problems, and to make it easy to use for non-experts. The progress from the past few years, and the large amount of applied research now being generated paints a positive future for MPC in practice. Together with this, deep theoretical work in MPC continues, ensuring that applied MPC solutions stand on strong scientific foundations.

References

1. Y. Aumann and Y. Lindell. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. In the *Journal of Cryptology*, 23(2):281-343, 2010 (extended abstract at *TCC 2007*).
2. D. Beaver, S. Micali and P. Rogaway. The Round Complexity of Secure Protocols. In 22*nd STOC*, pages 503–513, 1990.
3. M. Ben-Or, S. Goldwasser and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. 20*th STOC*, 1988.
4. Z. Beerliová-Trubíniová and M. Hirt. Perfectly-Secure MPC with Linear Communication Complexity. *TCC 2008*, Springer (LNCS 4948), pages 213–230, 2008.
5. R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
6. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In the 42*nd FOCS*, pages 136–145, 2001.
7. R. Canetti and A. Herzberg. Maintaining Security in the Presences of Transient Faults. In *CRYPTO'94*, Springer-Verlag (LNCS 839), pages 425–438, 1994.
8. R. Canetti, Y. Lindell, R. Ostrovsky and A. Sahai. Universally Composable Two-Party and Multi-Party Computation. In the 34*th STOC*, pages 494–503, 2002. Full version available at `http://eprint.iacr.org/2002/140`.
9. D. Chaum, C. Crépeau and I. Damgård. Multi-party Unconditionally Secure Protocols. In the 20*th STOC*, pages 11–19, 1988.
10. K. Chida, D. Genkin, K. Hamada, D. Ikarashi, R. Kikuchi, Y. Lindell and A. Nof. Fast Large-Scale Honest-Majority MPC for Malicious Adversaries. In *CRYPTO 2018*, Springer (LNCS 10993), pages 34–64, 2018.
11. R. Cleve. Limits on the Security of Coin Flips when Half the Processors are Faulty. In the 18*th STOC,* pages 364–369, 1986.
12. I. Damgård and J. Nielsen. Scalable and Unconditionally Secure Multiparty Computation. In *CRYPTO 2007*, Springer (LNCS 4622), pages 572–590, 2007.
13. I. Damgård, V. Pastro, N.P. Smart and S. Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In *CRYPTO 2012*, Springer (LNCS 7417), pages 643–662, 2012.
14. J. Doerner, Y. Kondi, E. Lee and a. shelat. Threshold ECDSA from ECDSA Assumptions: The Multiparty Case. In *IEEE Symposium on Security and Privacy 2019*, pages 1051–1066, 2019.
15. D. Evans, V. Kolesnikov and M. Rosulek. *A Pragmatic Introduction to Secure Multi-Party Computation*. NOW Publishers, 2018.
16. J. Furukawa and Y. Lindell. Two-Thirds Honest-Majority MPC for Malicious Adversaries at Almost the Cost of Semi-Honest. In the 26*th ACM CCS*, pages 1557–1571, 2019.
17. R. Gennaro and S. Goldfeder. Fast Multiparty Threshold ECDSA with Fast Trustless Setup. In the 25*th ACM CCS 2018*, pages 1179–1194, 2018.
18. O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In the 19*th STOC,* pages 218–229, 1987. Details in *Foundations of Cryptography: Volume 2 – Basic Applications* (Cambridge University Press 2004), by Oded Goldreich.
19. M. Hastings, B. Hemenway, D. Noble and S. Zdancewic. SoK: General Purpose Compilers for Secure Multi-Party Computation. In *IEEE Symposium on Security and Privacy 2019*, pages 1220–1237, 2019.

20. M. Ion, B. Kreuter, E. Nergiz, S. Patel, S. Saxena, K. Seth, D. Shanahan and M. Yung. Private Intersection-Sum Protocol with Applications to Attributing Aggregate Ad Conversions. *IACR Cryptology ePrint Archive*, report 2017:738, 2017.
21. Y. Ishai, J. Kilian, K. Nissim and E. Petrank. Extending Oblivious Transfers Efficiently. In *CRYPTO 2003*, Springer (LNCS 2729), pages 145–161, 2003.
22. Y. Ishai, M. Prabhakaran, A. Sahai. Founding Cryptography on Oblivious Transfer – Efficiently. In *CRYPTO 2008*, Springer (LNCS 5157), pages 572–591, 2008.
23. V. Kolesnikov, R. Kumaresan, M. Rosulek and N. Trieu. Efficient Batched Oblivious PRF with Applications to Private Set Intersection. In the 23*rd ACM CCS*, pages 818–829, 2016.
24. V. Kolesnikov and T. Schneider. Improved Garbled Circuit: Free XOR Gates and Applications. In *ICALP 2008*, Springer (LNCS 5126), pages 486–498, 2008.
25. A. Lapets, F. Jansen, K.D. Albab, R. Issa, L. Qin, M. Varia and A. Bestavros. Accessible Privacy-Preserving Web-Based Data Analysis for Assessing and Addressing Economic Inequalities. In *COMPASS 2018*, 48:1–48:5, 2018.
26. Y. Lindell. Fast Secure Two-Party ECDSA Signing. In *CRYPTO 2017*, Springer (LNCS 10402), pages 613–644, 2017.
27. Y. Lindell and A. Nof. Fast Secure Multiparty ECDSA with Practical Distributed Key Generation and Applications to Cryptocurrency Custody. In the 25*th ACM CCS*, pages 1837–1854, 2018.
28. Yehuda Lindell and Benny Pinkas. An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. In *EUROCRYPT*, pages 52–78. Springer, 2007.
29. J.B. Nielsen, P.S. Nordholt, C. Orlandi and S.S. Burra. A New Approach to Practical Active-Secure Two-Party Computation. In *CRYPTO 2012*, Springer (LNCS 7417), pages 681–700, 2012.
30. R. Ostrovsky and M. Yung. How to Withstand Mobile Virus Attacks. In 10*th PODC*, pages 51–59, 1991.
31. B. Pinkas, M. Rosulek, N. Trieu and A. Yanai. SpOT-Light: Lightweight Private Set Intersection from Sparse OT Extension. In *CRYPTO 2019*, Springer (LNCS 11694), pages 401–431, 2019.
32. B. Pinkas, T. Schneider and M. Zohner. Scalable Private Set Intersection Based on OT Extension. In *ACM Transactions on Privacy and Security*, 21(2):7:1–35, 2018.
33. T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multi-party Protocols with Honest Majority. In the 21*st STOC*, pages 73–85, 1989.
34. A. Shamir. How to Share a Secret. *CACM*, 22(11):612–613, 1979.
35. V. Shoup. Practical Threshold Signatures. In *EUROCRYPT 2000*, Springer (LNCS 1807), pages 207–220, 2000.
36. M. Vardi. The Long Game of Research. *CACM*, 62(9):7, 2019.
37. A. Yao. How to Generate and Exchange Secrets. 27*th FOCS*, pages 162–167, 1986.
38. Unbound Tech. (`www.unboundtech.com`), Sepior (`sepior.com`), and Curv (`www.curv.co`).
39. Sharemind, `https://sharemind.cyber.ee`.
40. Duality, `https://duality.cloud`.