2. Overview of Secure Multiparty Computation

MPC is a cryptographic tool that allows a collection of stakeholders to compute any function of their private inputs while maintaining the secrecy of each individual's input.[1] MPC protocols allow a collection of individuals to achieve anything that could be achieved in the presence of a trusted third party, but the trusted party is replaced by a transparent and provably secure cryptographic algorithm.

Large-scale public tests of MPC protocols have been performed in the case of secure auctions, where each bidder can be sure of the privacy of his bid, yet confident that the winning bidder was chosen correctly, and in secure elections, where each vote remains private but the tally is provably correct.[2] This type of secure auction does not require a trusted auctioneer; instead, the underlying cryptographic protocol ensures the privacy of the bids as well as the integrity of the auction result. In many settings, finding a trusted party (e.g., an auctioneer or ballot counter) can be difficult or impossible, and the scarcity of trusted parties allows those that do exist to charge a premium for their services. MPC protocols have the potential to bring the benefits of cooperation and coordination of operations into realms where it was previously impossible due to lack of trust.

Since its introduction, MPC has been a subject of intense study in the cryptographic community. Surveys of the MPC literature are available from Franklin and Yung (1996), Goldreich (2004), and Lindell and Pinkas (2009).[3] The potential benefits of MPC protocols have been widely recognized, but until recently most MPC protocols were too inefficient for practical use. Recent algorithmic advances, coupled with the steady increase in computing power, are beginning to make MPC efficient enough to be practical in a wide variety of settings. Currently, general software libraries for MPC exist that provide a high-level language (similar to Java or C), and code written in this language can, in principle, be compiled into secure implementations of any desired function. The FairPlay library was an initial attempt to provide a practical

[1] The MPC protocols we consider come with rigorous mathematical proofs that guarantee the privacy of each stakeholders' input.

[2] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, et al., *Multiparty Computation Goes Live*, Report 2008/068, IACR Cryptology ePrint Archive, 2008

[3] Matthew Franklin and Moti Yung, "Varieties of Secure Distributed Computing," in *Proceedings of Sequences II, Methods in Communications, Security and Computer Science,* 1996; Oded Goldreich, *Foundations of Cryptography,* Volume II, Cambridge University Press, 2004; Yehuda Lindell and Benny Pinkas, "Secure Multiparty Computation for Privacy-Preserving Data Mining," *The Journal of Privacy and Confidentiality*, Vol. 1, No. 1, 2009, pp. 59–98.

implementation of Yao's garbled circuits (discussed below).[4] For calculations involving three or more parties, libraries also exist implementing the MPC protocols of Ben-Or, Goldwasser, and Wigderson (also discussed below).[5]

To implement an MPC protocol, it is only necessary that each participant have a trusted computer on which to run his or her portion of the protocol and a (possibly insecure) way to communicate with the other participants. The protocol consists of a series of messages exchanged between the participants, at the end of which each participant learns the output of the protocol. The protocol itself is public, allowing each participant to independently verify that the software running on his or her own machine is valid. Additional cryptographic tools can be put in place to prevent participants from deviating from the prescribed protocol. Since each user only needs a computer (trusted by him or herself alone), and a connection to other users, MPC protocols can easily be implemented over the Internet.

Privacy Concerns Arising from MPC

Before delving into the details of how MPC protocols are implemented, we briefly outline two high-level privacy concerns that are inherent in any MPC protocol. In order to prove that a protocol is secure, a *threat model* needs to be introduced that formalizes the types of attacks that could be employed against the protocol. Once an MPC protocol has been proven secure in a given threat model, users have a strong guarantee that running the protocol *leaks no more information than the output of the protocol alone*, i.e., the protocol securely simulates a trusted third party. There are many situations, however, where the output of the protocol itself may leak too much information. For example, if two satellite operators securely compute a conjunction analysis and learn that there is a high probability of collision, then even if the MPC protocol is secure, each operator will have learned a lot of information about where the other's satellite is located. In the

---

[4] Dahlia Malkhi, Noan Nisan, Benny Pinkas, and Yaron Sella, "Fairplay - A Secure Two-Party Computation System," *USENIX Security Symposium '04,* 2004; Assaf Ben-David, Noam Nisan, and Benny Pinkas, "FairplayMP—A System for Secure Multi-Party Computation," in *CCS '08 Proceedings of the 15th ACM Conference on Computer and Communications Security*, New York: ACM, 2008, pp. 257–266.

[5] Bogdanov, D., S. Laur, and J. Willemson, "Sharemind: A Framework for Fast Privacy-Preserving Computations," In *Proceedings of the 13th European Symposium on Research in Computer Security: Computer Security*, ser. ESORICS '08, Vol. 5283, Berlin, Heidelberg: Springer-Verlag, 2008, pp. 192–206.; Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper B. Nielsen, "Asynchronous Multiparty Computation: Theory and Implementation," *Public Key Cryptography - PKC 2009, 12th International Conference on Practice and Theory in Public Key Cryptography, Irvine, CA, USA, March 18–20, 2009, Proceedings*, Springer, 2009, pp. 160–179; Seung Geol Choi, Kyung-Wook Hwang, Jonathan Katz, Tal Malkin, and Dan Rubenstein, *Secure Multi-Party Computation of Boolean Circuits with Applications to Privacy in On-Line Marketplaces*, IACR Cryptology ePrint Archive, Report 2011/257, 2011, pp. 416–432; Ben-David, Nisan, and Pinkas, 2008; Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson, "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation," in *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, Chicago, Ill., 1988, pp. 1–10.

satellite situation, this leakage seems to be acceptable to the community, but this is a question that needs to be addressed before any MPC protocol can be securely deployed.

Privacy Threat Models

Most MPC protocols consider one of three of the following threat models describing the behavior of the participants. In order of increasingly adversarial behavior, the three models are

1. honest-but-curious (semi-honest)
2. covert
3. malicious.

In the honest-but-curious or semi-honest model, all participants are assumed to follow all protocols correctly. If the protocol dictates that the participant should send a message of a specific form, the participant will send a message of that form. In this sense, participants are assumed to be honest. On the other hand, participants are also assumed to be curious, meaning that they will attempt to analyze any information or messages that they receive to glean information about the other participants' private information.

In the covert and malicious models, participants may behave arbitrarily. In particular, they may choose not to follow the protocol, and they may send other participants malformed messages in an attempt to learn other participants' private information. The difference between the covert and malicious security models is how often a cheating participant is caught. A protocol that is covert-secure is guaranteed to detect a participant that deviates from the protocol with some fixed probability (e.g., with 75 percent probability). A protocol that is secure in the malicious model (sometimes called fully secure) will essentially always[6] detect a participant that deviates from the protocol.

The honest-but-curious setting is not intended to successfully model real-world behavior; instead, it serves as the simplest model for designing protocols. Protocols that are secure in the honest-but-curious model can often be upgraded to protocols that are secure in the covert or malicious models using standard cryptographic techniques. The honest-but-curious model thus serves as a stepping-stone, and allows protocol designers to take a more modular approach to security design.

The covert model is intended to capture situations where the penalty for cheating is high relative to the potential gain. The covert security of the protocol, coupled with the high price for cheating, serves to prevent participants from deviating from the protocol.

The fully malicious model prevents cheating entirely. Protocols that are secure in the malicious model provide the strongest security guarantees, but are the hardest to design, and consequently they are the least efficient protocols in practice.

---

[6] Formally, cheating is detected with all but negligible probability, meaning that that probability that a participant can successfully cheat approaches zero faster than the inverse of any polynomial function of the security parameter.

As this report is concerned primarily with the feasibility of using MPC for conjunction analyses, we will focus attention on the simplest threat model, the honest-but-curious setting. As MPC protocols have never been developed for performing conjunction analyses, the honest-but-curious setting provides the natural starting point for exploration. If efficient protocols can be obtained in the honest-but-curious setting, these protocols can then be adapted to obtain the security levels necessary for real-world use.

Information Leakage in MPC

MPC is designed to eliminate the need for a trusted broker without sacrificing privacy. In many situations, however, when participants work together to calculate a function based on their private information, the output of the function may reveal private information even when the *calculation* of the function does not. For example, in the conjunction analysis setting, when two satellite operators give their private orbital information to a trusted third party to compute a conjunction analysis, if the trusted party says that a collision is likely, each operator gains information about the location of the other operator's satellite. This information leakage is inherent in the conjunction analysis calculation, because it occurs when there is a trusted third party and it occurs when the trusted third party is replaced by an MPC protocol.

MPC protocols leak no more information than a trusted third party would; nevertheless, information leakage can still be a problem. For example, a satellite operator could submit the orbital information of a fleet of hypothetical satellites to the trusted party in order to learn the locations of other participants' actual satellites.

This type of attack—submitting bogus orbital information—can be discouraged by calculating only whether the collision probability is above a certain threshold, by restricting the number of conjunction analyses any operator can perform, or by comparing each operator's inputs to the computation of the low-fidelity public orbital information and issuing a warning if there is a large discrepancy.

MPC protocols are designed to mimic the functionality of a trusted third party, so any information leakage that would occur in the presence of a trusted party will also occur in the MPC protocols. While these problems are not caused by MPC, whenever MPC is implemented in a new context, potential participants must decide *whether the output of the function alone* reveals too much private information. Tools exist to help potential participants analyze the amount of information that is revealed in this way.[7]

This report is concerned with the feasibility of using MPC for conjunction analyses and does not explore the amount of information revealed by the result of a conjunction analysis. If a framework for MPC were developed for conjunction analyses, potential

[7] P. Mardziel, M. Hicks, J. Katz, and M. Srivatsa, "Knowledge-Oriented Secure Multiparty Computation," *Proceedings of the 7th Workshop on Programming Languages and Analysis for Security*, ser. PLAS '12. New York: ACM, 2012.

participants would need to weigh the benefits of participation against the orbital information revealed by the output of conjunction analysis.

Converting Functions into Binary Circuits

This chapter provides an overview of the two major protocols for secure two-party computation; namely, Yao's garbled circuit and the Goldreich, Micali, and Wigderson (GMW) protocol.[8] Although these protocols differ significantly, both convert the function being computed into a binary circuit, and then provide a method for securely computing each gate of the circuit using a cryptographic protocol called Oblivious Transfer (OT).

A Boolean gate is a function with a one-bit output. It will be sufficient to consider gates with two single-bit inputs and one single-bit output. One of the simplest gates is an AND gate, which outputs zero unless both input bits are one, in which case it outputs one. The "not-and" or NAND gate reverses the output of an AND gate, outputting one unless both inputs are one.

Each gate, which has two binary input wires and one binary output wire, has an associated truth table that relates the input to the output. An example is shown in Table 2.1.

Table 2.1
Truth Table for a NAND Gate

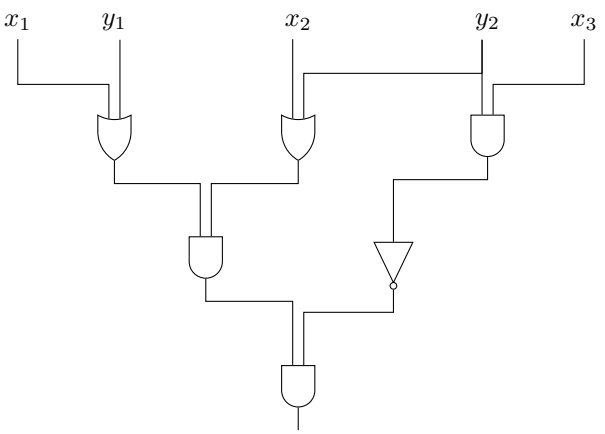
Input 1	Input 2	Output
0	0	1
0	1	1
1	0	1
1	1	0

Each truth table can be represented concisely using four bits of information, listing the outputs for each of the four possible inputs (this corresponds to the last column of Table 2.1). Simple Boolean gates can be combined to create more complex functions. Figure 2.1 shows an example of a Boolean circuit with six gates, of depth three,

[8] Andrew C. Yao, "How to Generate and Exchange Secrets," *27th Annual Symposium on Foundations of Computer Science (FOCS 1986),* Toronto, October 27–29, 1986, pp. 162–167; Oded Goldreich, Sylvio Micali, and Avi Wigderson, "How to Play ANY Mental Game," *STOC 1987: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, New York: ACM, January 1987, pp. 218–229. For an in-depth discussion of these protocols, see Carmit Hazay and Yehuda Lindell, *Efficient Secure Two-Party Protocols: Techniques and Constructions*, Springer, 2010.

computing the function $[(x_1 \vee y_1) \wedge (x_2 \vee y_2)] \wedge \neg(y_2 \wedge x_3)$.[9] While any function can be represented as a circuit, computing complex functions requires extremely large circuits. For example, a circuit that computes a single multiplication of floating-point numbers requires tens of thousands of binary gates.[10]

Figure 2.1
Example of a Boolean Circuit



Yao's Garbled Circuit

The first protocol for secure two-party computation that this report explores is *Yao's garbled circuit*. Andrew Yao introduced the notion of MPC[11] and outlined the first two-party secure computation protocol in 1986.[12] His work described how two parties could securely calculate any public function of their joint inputs by introducing a technique that came to be known as "circuit garbling." This section provides an overview of Yao's

[9] The symbols ∧, ∨, and ¬ denote the binary operations AND, OR, and NOT, respectively.

[10] Reza Hashemian, "A New Multiplier Using Wallace Structure and Carry Select Adder with Pipelining," *ISCAS '02 Conference Proceedings*, 2002.

[11] Andrew C. Yao, "Protocols for Secure Computations," *23rd Annual Symposium on Foundations of Computer Science* (FOCS 1982), Chicago, Ill., November 3–5, 1982, pp. 160–164; Yao, 1986.

[12] Yao, 1986.

construction, omitting many of the technical details needed for security.[13] The work of Lindell and Pinkas provides a rigorous technical description of the protocol.[14]

Yao's garbled circuit allows two participants, denoted A and B, to compute the function $f(a,b)$, where a denotes the private input of A and b denotes the private input of B. In the case of a conjunction analysis, each party's private input is the location and velocity of their satellite and the function being calculated is the function that outputs the probability of collision.[15]

The structure of Yao's protocol is fundamentally asymmetric; one party "garbles" the circuit, and the other evaluates the garbled circuit. Despite this asymmetry in the construction, the security guarantees and outputs of the protocol can be made symmetric.

Yao's protocol (see Figure 2.2) begins as follows. Party A will garble the circuit for f gate-by-gate. To garble the gate, party A will employ a symmetric-key cryptosystem E.[16] A symmetric key cryptosystem relies on a secret key, k, and provides the guarantee that if the key is unknown the encryption $E_k(s)$ provides no information about the secret, s. Yao's technique requires two encryption steps, creating a double encryption, by encrypting the secret, s, under two different keys. This provides the guarantee that no information about the secret is leaked unless both keys are known. To garble a gate, for each input wire and each output wire of the gate, party A chooses two uniformly random keys (see Figure 2.3). Party A then creates a garbled truth table, by creating four double encryptions (requiring four secret keys), as in Table 2.2. Party A then randomly shuffles the rows of the truth table. At the end of the garbling procedure, party A has two secret keys for each wire of the circuit, and a double-encrypted truth table for each gate of the circuit.

---

[13] Yao's original protocol, as described here, only provides security against passive adversaries; Lindell and Pinkas described an extension of Yao's protocol to provide security against active adversaries. See Yehuda Lindell and Benny Pinkas, "Secure Two-Party Computation via Cut-and-Choose Oblivious Transfer," in Y. Ishai, ed., *Theory of Cryptography*, Vol. 6597 of *Lecture Notes in Computer Science*, Berlin, Heidelberg: Springer, 2011, pp. 329–346.

[14] Yehuda Lindell and Benny Pinkas, *A Proof of Security of Yao's Protocol for Two-Party Computation*, ePrint 2004/175, 2004.

[15] For example, in the case of an auction, inputs are the private values, a and b, and the function being computed is essentially the function that computes the maximum of those inputs.

[16] In practice this is a system like the Advanced Encryption Standard (AES). In the case of AES-256 a secret key is just a uniformly random 256-bit string.

Figure 2.2
Yao's Garbled Circuit Protocol

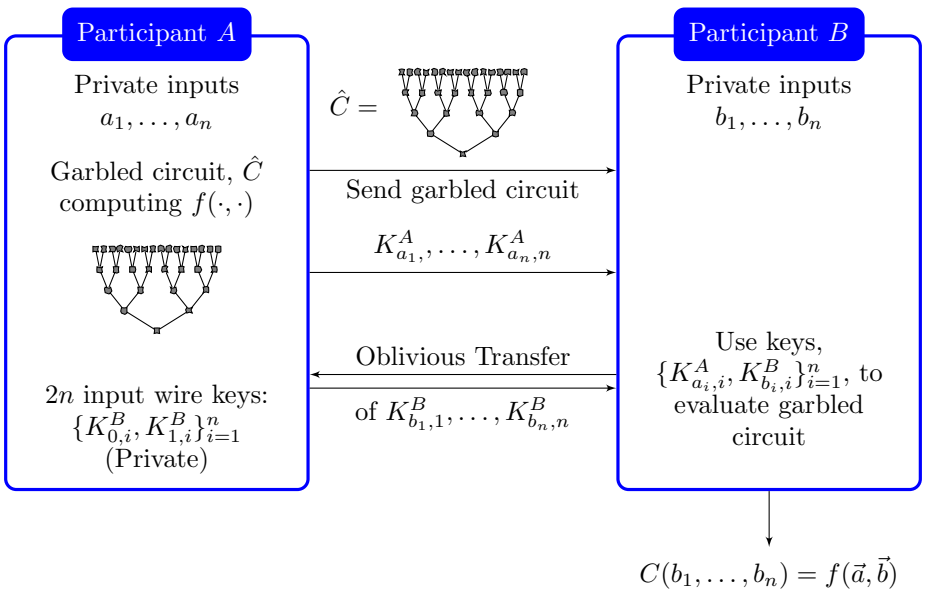


Figure 2.3
Garbled Gate

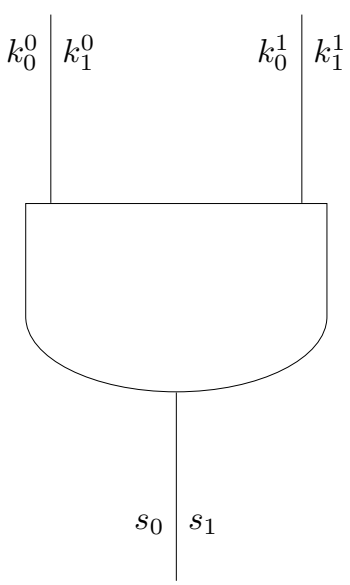


Table 2.2
Garbled Truth Table for a NAND Gate

Input 1	Input 2	Plaintext Output	Garbled Output
0	0	1	$E_{k_0^0}(E_{k_0^1}(s_1))$
0	1	1	$E_{k_0^0}(E_{k_1^1}(s_1))$
1	0	1	$E_{k_1^0}(E_{k_0^1}(s_1))$
1	1	0	$E_{k_1^0}(E_{k_1^1}(s_0))$

The important observation is that given one key from each input wire, exactly one of the four encrypted outputs can be decrypted. The values s_0 and s_1 are the two keys corresponding to the output wire of the garbled gate. The fact that the first three rows encrypt s_1 while the last row encrypts s_0 makes this a garbling of a NAND gate. In this example, given k_1^0 and k_1^1 corresponding to inputs 1, 1, the ciphertext $\mathrm{E}_{k_1^0}(E_{k_1^1}(s_1))$ can be decrypted, and the key s_1 can be recovered (but not the fact that it corresponds to the value one). Thus having only two keys, k_1^0 and k_1^1, reveals s_1, but nothing about what type of gate was garbled. The output keys for this gate, s_0, s_1, will then be used as the keys on the input wire for the next gate in the circuit. In this way, party A will create a garbling of the entire circuit for the function f. At the end of this process, party A has created two keys for each wire and a garbled truth table for each gate. For the final gates of the circuit (the output gates), party A creates encryptions of the actual (binary) gate output instead of secret keys—i.e., for output gates $s_i = i$. Then, party A gives the entire garbled circuit (consisting of all the garbled truth tables, but not the keys) to party B. For each gate whose input wires come from party A's input, party A gives B the key to that wire corresponding to her input bit. Since this key is a uniformly random string, these keys reveal nothing about party A's input to B.

Party B now has the garbled circuit. Each wire that corresponds to one of B's inputs has two secret keys associated with it. One key is associated to an input of 0 and the other is associated to an input value of 1. To compute the circuit, party B needs the keys corresponding to his input bits on each of these input wires. At this point, party A knows both keys for each wire, but cannot reveal them both because this would reveal the entire circuit and hence A's private input. Party B knows which one out of each pair of keys that he needs, but he cannot simply reveal which key he needs because this is exactly his private input information. To allow B to acquire the necessary keys from party A, the two parties engage in an oblivious-transfer (OT) protocol (detailed below). For the input wires corresponding to party B's inputs, OT guarantees that party A does not learn party B's input bit (necessary for B's privacy) and party B learns only the key corresponding to his input bit (and not the other key). Once party B has one key (out of the pair of keys) for each input wire, he can use the garbled truth tables to compute the keys for the

next level. Proceeding in this way, party B can compute the entire circuit. When B has evaluated the entire garbled circuit, he will have learned a single entry in the garbled truth tables of each output gate. By design, these values correspond to the output bits of the circuit evaluated on A and B's inputs.

This protocol requires that party A send the entire garbled circuit, and one key for each of her input wires, as well as one OT for each of party B's input bits. The remainder of the protocol does not require communication between the parties. Since Yao's original work, many significant efficiency improvements have been made.[17]

Yao's protocol only provides a method for secure two-party computation. This means that Yao's technique is not suitable for solving problems that inherently involve many parties (e.g., auctions, elections). Although there may be many satellite operators, each conjunction analysis involves only two satellites, and hence two-party computation is the appropriate model for securely computing conjunction analyses.

The Goldreich, Micali, and Wigderson Protocol

The second protocol for MPC that this report explores was developed by Goldreich, Micali, and Wigderson (called the "GMW protocol").[18] The GMW protocol provides an alternative to Yao's protocol for securely computing a function. Conceptually, the GMW protocol is very different from Yao's protocol. The GMW protocol is much more symmetric, and the underlying protocol easily extends to handle an arbitrary number of parties, whereas Yao's technique is only applicable in the two-party setting.

The GMW protocol is built on secret sharing.[19] Secret sharing is a means of distributing a secret among a number of parties, so that each party individually has no information about the secret, but together they can recover the secret.

The primary idea of the GMW protocol is to distribute each party's input using a secret-sharing scheme. Each party has a share of every party's secret input. The core of the GMW protocol is a method that allows the parties to perform a computation on the input shares in such a way that at the end of the protocol, each party is left with a share of the output.

As in Yao's protocol, the GMW protocol works on a gate-by-gate basis. The GMW protocol over the binary field proceeds as follows. Two parties, denoted A and B, with

---

[17] V. Kolesnikov and T. Schneider, "Improved Garbled Circuit: Free XOR Gates and Applications," *Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Part II*, ICALP '08, Berlin, Heidelberg: Springer-Verlag, 2008, pp. 486–498; B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams, "Secure Two-Party Computation Is Practical," *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '09, Berlin, Heidelberg: Springer-Verlag, 2009, pp. 250–267..

[18] Goldreich, Micali, and Wigderson, 1987.

[19] Adi Shamir, "How to Share a Secret," *Communications of the ACM*, Vol. 22, No. 11, November 1979, pp. 612–613.

inputs a and b, begin by secret-sharing their inputs. To secretly share her input, party A chooses a random a_1, a_2 subject to the constraint $a_1 + a_2 = a$, and gives a_2 to B.[20] Similarly, party B chooses b_1, b_2 subject to the constraint that $b_1 + b_2 = b$, and gives b_1 to party *A*. The numbers a_1, a_2 are called shares of a. At this point A has (a_1, b_1), and B has (a_2, b_2). The share a_2 is independent of a, so B learns nothing about A's input, and vice-versa. The two parties, A and B can now perform computations on the shares as follows:

- Addition Gates: Each party starts with a share of the secret a and a share of the secret b and their goal is to end up with a share of the secret $a + b$. To do this, each party simply adds its two shares. Party A is left with $a_1 + b_1$, and party B is left with $a_2 + b_2$. Since $a + b = (a_1 + a_2) + (b_1 + b_2) = (a_1 + b_1) + (a_2 + b_2)$, each participant is left with a valid share of the sum $a + b$.
- Multiplication Gates: Each party starts with a share of the secret a and a share of the secret b and their goal is to end up with a share of the secret ab. Unlike the case of addition gates, computing multiplication gates cannot be done without communication between the participants. The goal in computing a multiplication gate, is to end up in a situation where participant A has a share c_1, and participant B has a share c_2 such that $c_1 + c_2 = ab$, and c_1, c_2 are uniformly random (but not independent). To accomplish this, party A chooses $c_1 \in \{0,1\}$ uniformly at random. The protocol will be successful if party B is left with the share $ab + c_1 = (a_1 + a_2)(b_1 + b_2) + c_1$. This is accomplished as follows. Party A computes the four values
  $(s_1, s_2, s_3, s_4) = (c_1 + a_1 b_1, c_1 + a_1(b_1 + 1), c_1 + (a_1 + 1)b_1, c_1 + (a_1 + 1)(b_1 + 1))$
  corresponding to the four possible values of B's shares a_2, b_2. If party B can select the correct s_i corresponding to his shares, the protocol will succeed. This is accomplished using one-out-of-four OT, with party A acting as a sender with inputs (s_1, s_2, s_3, s_4), and party B acting as receiver with input $1 + 2a_2 + b_2$. This protocol allows B to learn $c_1 + (a_1 + a_2)(b_1 + b_2)$, which will be his share of the product ab.

By performing the above actions for each gate of the function being computed, the parties will end up with shares of the output of the function. These shares can then be combined to reveal the output of the function.

Like Yao's protocol, the security of the GMW protocol rests on the security of the underlying OT, and both Yao's protocol and the GMW protocol are proven to be secure, assuming the existence of a secure implementation of OT. OT (described in detail below)

[20] Throughout this section, we use arithmetic in the binary field. So addition corresponds to the XOR operation on bits and multiplication corresponds to the AND operation on bits.

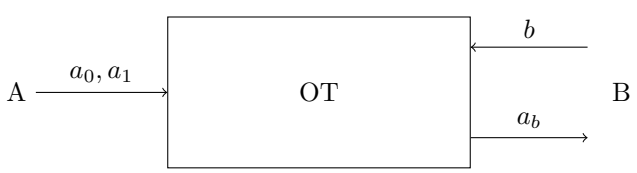
is a conceptually simple cryptographic primitive, and there are many known provably secure implementations of OT, any one of which could be used to construct MPC protocols. Although both Yao's protocol and the GMW protocol rely on OT, there are some fundamental differences between the two protocols. In Yao's protocol, all of the OTs can be computed in parallel at the beginning of the communication, and the number of OTs necessary is proportional to B's input size.[21] In the GMW protocol, since each multiplication gate requires an OT, the number of rounds of communication is proportional to the depth of the circuit, and the number of OTs is proportional to the number of multiplication gates in the circuit instead of the input size.[22] Whether Yao's protocol or the GMW protocol is more efficient will depend on the type of function being evaluated.

Because Yao's protocol is a two-round protocol, and all the OTs needed in the protocol can be executed in parallel, Yao's protocol is less sensitive to the effects of network latency. The GMW protocol, on the other hand, requires a number of rounds of interaction between the participants that is proportional to the depth of the circuit being evaluated. In a multi-round protocol like GMW, the computation required in each round of the protocol cannot be started until all of the previous round's messages have been received. This means that, even if the total computation required in the GMW protocol is small, the GMW protocol may be less suitable for situations where the communication latency is high.

Oblivious Transfer

OT is a two-party protocol between a sender and a receiver, illustrated in Figure 2.4.

Figure 2.4
One-Out-of-Two OT



The sender, A, has inputs a_0, a_1, and the receiver, B, has a choice bit b. At the end of the protocol, B should learn the chosen input a_b. The protocol is secure if two conditions are satisfied: (1) the sender, A, does not learn the receiver's choice bit b, and

[21] Yao's protocol requires "string OTs," where the sender has two secret strings, and the receiver receives one of them.

[22] The GMW protocol uses "bit OTs," where the sender has two secret bits, and the receiver receives one of them.

(2) the receiver, B, does not learn the sender's other input a_{1-b}. Many variants of OT exist, and in particular, the GMW protocol described here requires one-out-of-four OT, where the sender has inputs, and the receiver learns one of them.

The GMW protocol requires performing OTs for each gate of the circuit being evaluated.[23] This can result in millions of OT evaluations to securely compute even fairly simple functions. Thus the efficiency of the underlying OT protocol plays a large role in determining the efficiency of the overall MPC protocol. It has been the recent improvements in the efficiency of OT that have led to major efficiency improvements in MPC.[24]

Computing an OT protocol requires two-way communication between the sender and the receiver, and requires both parties to perform (possibly expensive) cryptographic computations. To improve the online efficiency of any MPC protocol based on OT, OTs can be precomputed. Precomputing OTs is useful in scenarios where the parties know they will want to perform a calculation in the future, but do not yet know the data on which they will perform the calculation. For example, two satellite operators who know that they will want to perform a conjunction analysis tomorrow can perform all the OT calculations today. This technique cannot decrease the total amount of time necessary for the computation, but it can drastically reduce the amount of time between when the inputs are learned and when the computation finishes.

To precompute an OT, the sender picks random inputs r_0, r_1, the receiver chooses a random choice bit t, and they engage in the standard OT protocol, leaving the receiver with r_t. At a later time, to perform an OT on inputs a_0, a_1 with choice bit b, the receiver sends $t + b$, and the sender responds with $a_0 + r_{t+b}, a_1 + r_{1-(t+b)}$.[25] The receiver can recover a_b by subtracting the known value r_t. Thus, after having precomputed a random OT, any OT can be performed using only three additional bits of communication and no cryptographic calculations. By precomputing an OT, the parties can later run the OT using only three bits of communication.[26] When performing millions of OT protocols, this can result in significant savings.

---

[23] The GMW protocol also requires a number of rounds equal to the depth of the circuit. Because each round requires communication between the parties, if network latency is high, it may be prohibitively slow to have too many rounds of communication.

[24] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank, "Extending Oblivious Transfers Efficiently," *CRYPTO*, 2003, pp. 145–161.

[25] Where all computations are done modulo 2, i.e., in the binary field where addition corresponds to the XOR operation on bits and multiplication corresponds to the AND operation on bits.

[26] Precomputing OTs can significantly reduce the amount of online computation and communication necessary in an MPC protocol, but it cannot reduce the number of rounds of communication. Thus, if the performance bottleneck is caused by network latency, precomputing OTs will have little benefit. In practice, however, it seems that computation time and network bandwidth are often the limiting factors in performance, and in these situations precomputing OTs can be beneficial.

Multiparty Computation with More Than Two Participants

MPC with more than two parties has also been studied, and many protocols have been developed, but they can all be seen as variants of the original schemes of Ben-Or, Goldwasser, and Wigderson (BGW) and Chaum, Crépeau, and Damgård (CCD).[27]

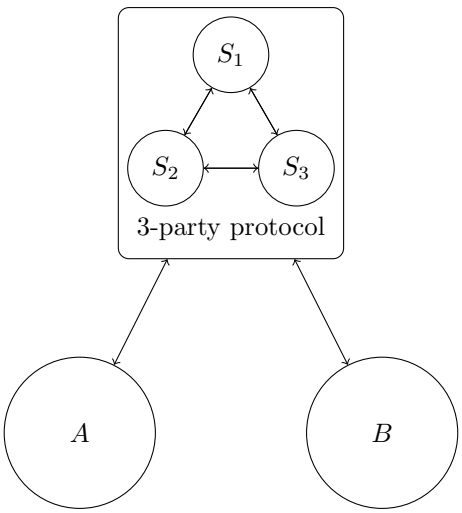
The BGW protocol can guarantee unconditional security in the case that the majority of participants follow the protocol honestly. These protocols are often called "honest majority" protocols. Unlike Yao's protocol and the GMW protocol, the BGW and CCD protocols do not use OT, and hence their efficiency is not affected by the speed of OT protocols.

In the case of a two-party computation, however, the BGW and CCD protocols cannot guarantee any type of security. A conjunction analysis is a two-party calculation, however, so the BGW and CCD protocols are not immediately applicable here. Although there may be many operators maintaining satellites, and any individual operator may wish to perform many different conjunction analyses simultaneously, but each conjunction analysis calculation remains a calculation between two parties. Thus, a two-party MPC protocol is required. There are other settings, however, where the calculations necessarily involve more than two parties. An example of a truly multiparty problem would be an auction or an election. The winner of an election, for example, cannot be computed via a series of pairwise calculations without revealing excess information. Similarly, trying to compute the highest bidder in an auction via pairwise calculations would reveal the higher bidder from every pair of bidders when only the highest bidder of the entire group needs to be revealed.

Although honest-majority MPC protocols like BGW have more limited applicability than protocols like GMW, honest-majority protocols can often be computationally more efficient than two-party protocols. To capitalize on this performance advantage, it is common to convert a two-party protocol into a three-party protocol in the following manner. If two parties, A and B wish to perform a two-party calculation, they can employ three servers S_1, S_2, S_3 and secret-share their data among the three servers. The three servers can run the secure three-party protocol and return the answer back to the original participants. This scenario is described in Figure 2.5.

[27] Ben-Or, Goldwasser, and Wigderson, 1988; David Chaum, Claude Crépeau, and Ivan Damgård, "Multiparty Unconditionally Secure Protocols," *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, Chicago, Ill., 1988, pp. 11–19.

Figure 2.5
Converting a Two-Party Protocol into a Three-Party Protocol



Since the three-party protocol will provide security if the majority of servers are honest, this modified protocol will provide security to the two participants if at least two out of three of the servers perform the protocol correctly. If the two parties knew exactly which server was honest, no MPC protocol would be necessary; the honest server could simply act as the trusted third party and calculate the function alone.[28]

This method of converting a three-party solution to a two-party solution can be made more computationally efficient, and hence three-party solutions have frequently been proposed for two-party problems where the two-party solution cannot be made efficient enough. The three-party solution reduces the amount of trust necessary, but does not eliminate it completely.

Requirements for Implementing MPC

Current MPC protocols are implemented as software systems. Each participant in the protocol receives an identical piece of software implementing the MPC protocol. The security of the system does not rely on any hidden aspects of the software, and hence there are no security issues involved in providing each participant with an independent copy of the software. Each client must have a trusted computer system on which to run an MPC protocol. Each client's trusted hardware will feed the client's secret inputs to the MPC software running on the system.

[28] Alternatively, the two parties could employ a single outside server, and the computation could be guaranteed secure as long as each party trusted the other party *or* the outside server to behave honestly.

The privacy of the protocol hinges on the fact that the randomness generated by each individual participant cannot be learned or influenced by any other participant. Generating high-quality randomness on a computer can be difficult, but it is essential to the security of the protocol, because if any participant's random choices can be predicted, the privacy of the protocol is compromised. Finally, the individual MPC software systems must be able to communicate. This is accomplished by establishing communication channels between each participant's trusted computer systems. A standard network connection suffices for this purpose, and all communication across these channels will be dictated by the MPC software implementation. The speed of the connection (both the bandwidth and the latency) will affect the speed of the MPC.

To calculate complex functions, the calculations required by each participant and the information communicated between participants can be quite large. Increased processing power will increase the speed of the local calculations, and faster data links between the participants will increase the speed of the communication. Whether the computation or communication provides the performance bottleneck will depend on the function being calculated, along with the specific computing infrastructure. In most cases, however, when performing multiparty protocols using traditional data links (e.g., the Internet) the communication provides the bottleneck and efficiency can be significantly improved by providing faster connections between the participants.

In certain situations, to maximize the speed of the network connection, it may be necessary to house each participant's trusted hardware in the same physical location. For example, a single building could be provided such that each participant has his or her own secure area where their computer systems are located. Recent tests indicate a slowdown of 17–64 percent when performing secure computations over the Internet instead of a local area network (LAN).[29]

The security of Yao's protocol and the GMW protocol rests on OT, and in the GMW protocol, performing these OTs comprises the bulk of the communication required between participants. Since OTs can be precomputed in any protocol based on OT (e.g., Yao, GMW), the participants in the MPC protocol can be constantly precomputing OT pairs. These precomputed OT pairs can then be used to securely and quickly perform the necessary secure calculations as they arise, allowing the participants to quickly perform time-sensitive calculations.

Thus far, we have discussed the protocols and requirements for MPC in general. In the following chapter, we describe how such capabilities could be applied to orbital conjunction analysis.

[29] Choi et al., 2011.