# A Review of Multi-Party Computing Primitives

Qiantan Hong, Daniel Liu, Jeffery Yu

May 20, 2021

**Abstract**

Multi-party computation (MPC) is a cryptographic idea that allows multiple parties to do computation with each party's data, without revealing that data to other parties. In this paper, we collate several fundamental, important, or interesting MPC primitives, explain the algorithms to achieve them, as well as present a new perspective unifying many of these primitives together.

## Contents

# 1 Introduction

The goal of secure multi-party computation (MPC) is to study methods to compute functions with inputs supplied by involved parties, yet keeping those parties' inputs private from each other. A classic example of this is "Yao's millionaires problem" [12], in which two millionaires wish to determine who is richer without revealing their wealth. Of course, the poorer person will learn that the richer person has at least as much money as them and vice-versa, so it is not zero-knowledge, but rather no information is revealed beyond the answer.

Beyond just hypothetical examples of determining who is richer, there are also practical applications of MPC. A large real-world example was implemented in a secure, private auction in 2008 [2]. A very relevant example today is contact tracing, in which each person wishes to know if any of their close contacts have tested positive for COVID-19, without discovering who is positive. Many cities employ a team of contact tracers who serve as an intermediate between a positive patient and their close contacts.

This resembles the *ideal world model*, in which there exists an incorruptible, trusted party who will receive the private inputs and perform the computation, revealing only the final answer. This however still requires divulging private information to a third party, which individuals may not desire. Rather, MPC protocols typically work with the *semi-honest model*, in which all parties are assumed to honestly follow the protocol agreed upon, but will use any information gained to deduce information about other parties [3].

**Definition 1.** An interactive protocol $\pi$ that computes a function $f(x_1, \ldots, x_n)$ is *secure against semi-honest adversaries* if there exist probabilistic polynomial time simulators $S_1, \ldots, S_n$ such that for each $i$, the simulator's transcript is computationally indistinguishable from the view of the $i$th party:

$$S_i(x_i, f(x_1, \ldots, x_n)) \cong_c \text{View}_i(\pi(x_1, \ldots, x_n))$$

for all $x_1, \ldots, x_n \in \{0,1\}^k$.

In other words, no party learns any information beyond their private input and the public output. The semi-honest model is a version of passive security. In particular, it has no protection against malicious adversaries who actively provide false data or deviate from the agreed protocol. In all of the algorithms we discuss below, there are many opportunities for parties to lie to gain additional information. The semi-honest model precludes that. This is nevertheless a reasonable security model, as a group of friends may trust each other to honestly follow the protocol in order to obtain the answer, while at the same time wanting to keep their own information private.

In [11], Yao presents a way for two parties to securely evaluate a function in terms of boolean circuits, known as Yao's garbled circuit. Theoretically, this can encompass all reasonable algorithms. However, it can be hard to convert arbitrary functions to boolean circuits, and the boolean circuit itself may be costly in computation time or communication time. Therefore, it remains an interesting and relevant problem to find efficient secure protocols for specific algorithms. In this paper, we will focus on two-party computation protocols.

# 2 Background

Before we dive into specific primitives, it is first worth broadly explaining the background behind Yao's garbled circuit, as it both is historically important and motivates the general idea of how MPC can be achieved.

## 2.1 1-2 Oblivious Transfer (OT)

The 1-2 oblivious transfer (or 1 out of 2 oblivious transfer, or OT for short) is a protocol that allows two parties Alice and Bob to send a chosen message without revealing any extra information. Specifically, Alice has two messages $m_0, m_1$, and 1-2 oblivious transfer allows Bob to learn one of these messages without learning the other and without Alice learning which message Bob chose.

To set up this protocol, we will follow the construction supplied by [6]. Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public key cryptographic protocol with plaintext space $\mathcal{M}$. For our example, we will use RSA public key cryptography. Recall that $\mathsf{Gen}$ outputs a public key $(n, e)$ where $n = pq$ is the product of two large primes and $e$ is an encryption exponent, as well as a secret key $d$. Encryption and decryption are defined as

$$\mathsf{Enc}(\mathsf{pk}, m) = m^e \pmod{n},$$

$$\mathsf{Dec}(\mathsf{sk}, c) = c^d \pmod{n}.$$

Thus, $d$ should be chosen so that the scheme is correct.

For the actual scheme, we follow these steps:

1. Alice creates two messages $m_0, m_1$ as well as a instance of RSA, and publishes the public key $(n, e)$. Alice also publishes two random plaintexts $r_0, r_1 \in \mathcal{M}$.
2. Bob chooses a $b \in \{0, 1\}$ corresponding to which message he would like, as well as a random plaintext pad $s \in \mathcal{M}$. Bob sends over $\mathsf{Enc}(\mathsf{pk}, s) + r_b$.
3. Alice computes
$$t_0 = \mathsf{Dec}(\mathsf{sk}, (\mathsf{Enc}(\mathsf{pk}, s) + r_b) - r_0),$$
$$t_1 = \mathsf{Dec}(\mathsf{sk}, (\mathsf{Enc}(\mathsf{pk}, s) + r_b) - r_1).$$
$t_b$ will equal $s$, while the other produces gibberish, but Alice doesn't know which.
4. Alice sends back $t_0 + m_0$ and $t_1 + m_1$.
5. Bob calculates $m_b = t_b + m_b - s$, since $t_b = s$. Bob cannot deduce any more information because $t_{1-b}$ is gibberish to him since it uses the $\mathsf{Dec}$ function.

## 2.2 Yao's Garbled Logic Gates

OT provides the fundamental primitive needed to construct the key ingredient of Yao's garbled circuits: garbled logic gates. Alice and Bob each have a bit, and the game is for them to try to find the evaluation of these bits on a logic gate without revealing any extra information about the other person's bit. If we can construct such a protocol for a set of universal logic gates, then the immediate consequence is that we can find the output of any function taking two binary inputs from Alice and Bob respectively without leaking any extra information, as a set of universal logic gates by definition can construct any function. This is the idea behind *Yao's garbled circuit.* [11]

Since $\mathsf{NOR}$ by itself is a universal logic gate, we will construct this gate in a privacy preserving manner. In fact, the following construction works for any binary input gate.

The protocol consists of two jobs: Alice, the Garbler, and Bob, the Evaluator.

1. They start with a table showing the inputs and outputs of the $\mathsf{NOR}$ gate:

| | 0 | 1 |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 0 | 0 |

2. Without loss of generality, assume Alice's bit determines the row and Bob's the column. Alice then replaces her bits, Bob's bits, and the output bits with random "garbled" versions:

| | $A_0$ | $A_1$ |
|---|---|---|
| $B_0$ | $C_1$ | $C_0$ |
| $B_1$ | $C_0$ | $C_0$ |

where $A_0, A_1, B_0, B_1, C_0, C_1 \in \mathcal{M}$. Bob doesn't know which of these correspond to 0 and which correspond to 1.

3. Alice then creates an encryption algorithm $(\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ public to everyone that takes as seed to $\mathsf{Gen}'$ a pair of messages in $\mathcal{M}$. For $A_a, B_b$, she generates a key $\mathsf{sk}_{A_a,B_b}$ and uses it to encrypt the corresponding output bit:
$$\begin{cases} \mathsf{Enc}'(\mathsf{sk}_{A_0,B_0}, C_1) \\ \mathsf{Enc}'(\mathsf{sk}_{A_0,B_1}, C_0) \\ \mathsf{Enc}'(\mathsf{sk}_{A_1,B_0}, C_0) \\ \mathsf{Enc}'(\mathsf{sk}_{A_1,B_1}, C_0) \end{cases}$$
She scrambles the four entries to make it not obvious which line corresponds to which output square.

4. The stage is now set for Bob to evaluate the circuit. Bob first asks Alice to give him the garbled value $A_a$ corresponding to Alice's bit $a$.

5. Bob then asks Alice to give him the garbled value $B_b$ corresponding to Bob's bit $b$. However, since Alice doesn't know $b$, Bob must ask for it with OT.

6. Finally, Bob computes $\mathsf{sk}_{A_a,B_b}$ and decrypts each of the four table entries. Assuming that $(\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ is set up such that trying to decrypt a ciphertext with the wrong key will give an invalid output $\perp$, exactly one of the four entries will give a valid decryption, and that decryption will exactly the garbled version of the output bit of the $\mathsf{NOR}$ gate.

As a note, the above protocol is for gates where Bob has an input. When Bob doesn't have an input, OT isn't even needed!

In any case, given any arbitrary function, Alice and Bob can dissect it into $\mathsf{NOR}$ gates, and apply the protocol for each gate to arrive at a final set of garbled bits as output. Through some commitment scheme, Bob can reveal these garbled bits to Alice and Alice can reveal the true values behind them, giving them both the function output without either knowing any extra information about the other's input.

Here, we see a main idea of two-party computation: asymmetry of roles. As we will find out, this theme will persist in the solution of many MPC primitives.

# 3 Primitives

We have seen that garbled circuits are able to implement a private version of any function. For simple functions that can be easily enumerated by a table of inputs and outputs, or are easily represented by a circuit, Yao's method works well. However, in practice converting arbitrary functions to circuits is often difficult and inefficient, especially more elaborate algorithms. For example, the problem of finding the shortest distance between two points on an arbitrary graph is not easily translated into a circuit, but there are other efficient algorithms for it [3]. In general, a direct garbling approach often involves directly enumerating all possible inputs. Therefore, it remains an interesting and problem to find efficient privacy-preserving algorithms for more complicated problems.

In the previous section we demonstrated OT as a primitive used in Yao's garbled circuits. Now we will present a collection of additional primitives found in in the literature. There is no universal definition of what constitutes a *primitive*. For the purpose of this work we mean a subroutine or building block that is frequently used in larger algorithms. We begin with several basic primitives, many of which can be efficiently solved directly via garbled circuits, and then present some more involved primitives where garbled circuits are less efficient.

## 3.1 Integer Comparison

Yao's millionaire problem is a specific case of the following general problem.

**Problem 1.** Given two $n$-bit integers $a$ and $b$, determine whether $a < b$, $a > b$, or $a = b$.

Yao's original solution in [12] is extremely inefficient, with a communication complexity that grows exponentially in the number of bits. However, that was four years before Yao published his garbled circuits.

Indeed, this problem is well-suited for garbled circuits. Specifically, the problem of comparing two $n$-bit integers is precisely solved with a digital comaparator circuit. The number of gates required for an $n$-bit comparator grows as $O(n)$.

## 3.2 Minimum and Maximum

**Problem 2.** Given two $n$ bit integers $a$ and $b$, determine $\min(a, b)$ (resp. $\max(a, b)$).

This problem can similarly be solved with a garbled circuit. Starting with a digital comparator, the circuit can be extended to return the minimum (resp. maximum) of the numbers. The number of gates here also grows as $O(n)$, so this is an efficient solution.

## 3.3 Bitwise OR

**Problem 3.** Given two $n$ bit sequence $a$ and $b$, determine $a \vee b$.

First, we observe that not only this problem can be solved with the garbled circuit algorithm, but this solution is also efficient. This is because we can perform OR on each bit independently, and each bit requires only one gate operation, therefore only 1-out-of-2 oblivious transfer.

Alternatively, we can compute bitwise OR using a semantically secure homomorphic encryption scheme. For example, based on ElGamal scheme, we construct the following protocol:

1. Alice generates their secret key $k$ and public key $q, g, g^k$ as in standard ElGamal scheme.
2. Alice sends to Bob ciphertext $c_{An} = (g^r, g^{a_n} \cdot g^{kr})$.
3. Upon receipt of $c_A n = (\alpha_n, \beta_n)$, Bob randomly pick $r'_n$, and send to Alice $c_{Bn} = (\alpha_n^{r'_n}, g^{b_n \cdot r'} \cdot \beta_n^{r'_n})$.
4. Alice decrypts $c_B n$ using their private key. Alice declares the $n$-th bit of the result to be 0 iff $c_B n$ decrypts to 1.

The secrecy of $a$ follows from the semantic security of the homomorphic encryption scheme. Suppose $a_n = 1$, the secrecy of $b_n$ then follows from the same semantic security, which prevents Alice from distinguishing $g^{r'}$ and $g^{2r'}$ without knowing $r'$.

## 3.4 Set Union

**Problem 4.** Given two sets $A$ and $B$, determine $A \cup B$.

The privacy aspect of this is that $A \cap B$ remains hidden to both parties. That is, if Alice sees an element in $e \in A \cup B$ not in $A$, she knows $e \in B$, but if she sees an element $e \in A$, then she does not know whether it is in $B$ as well.

There are many privacy-preserving algorithms for set union, such as those presented in [3]. All of these assume a finite universe $\mathcal{U}$ of possible elements. Since the universe is finite, parties can agree on a canonical ordering of the elements. The efficient algorithm we present will use the privacy-preserving min function as a primitive.

1. Alice and Bob agree on an ordering of the universe $\mathcal{U}$. Also agree on some representation of $\infty$ larger than any element in the universe.
2. Initialize $S = \varnothing$.
3. Alice selects the minimal element $a \in A$, and Bob the minimal element $b \in B$, or $\infty$ if their set is empty.
4. They apply the privacy-preserving protocol for $\min(a, b)$ and append that to $S$.
5. If $\min(a, b) \in A$, Alice removes it from her set and chooses the next minimal element. Similarly, if Bob's set contains $\min(a, b)$, he removes it and moves to the next minimal element.

6. Repeat steps 3–5 until $\min(a, b) = \infty$, at which point $S = A \cup B$.

This has communication and computational complexity $O(|S| \lg |\mathcal{U}|)$, since each element can be represented in $\lg |\mathcal{U}|$ bits, each iteration has complexity $\lg |\mathcal{U}|$ from the privacy-preserving minimum, and there are $|S|$ total iterations.

We note that this is an instance where we have an algorithm that is faster and more efficient than directly using circuits (even though we do indirectly use circuits through the min primitive). In particular, a direct garbled circuit/table approach involves $|\mathcal{U}|$-bit inputs for all possible subsets of the universe, which will take at least $O(|\mathcal{U}|)$ gates. For very large universes of elements, where $|S| \ll |\mathcal{U}|$, we have $|S| \lg |\mathcal{U}| \ll |\mathcal{U}|$, so the algorithm using the min primitive outperforms the direct-garbling approach.

## 3.5 Set Intersection

**Problem 5.** Given two sets $A$ and $B$, determine $A \cap B$.

The privacy aspect of this is that the symmetric difference $A \Delta B$ remains hidden to both parties. That is, each party knows that the elements in $A \cap B$ are common to both parties, but they do not know the other elements in the other party's set.

There are also many privacy-preserving algorithms for set intersection, and this remains an active problem of research [8]. As with set union we consider a large but finite universe of possible elements $\mathcal{U}$. We present the standard, "naive" algorithm:

1. Alice and Bob agree on a cryptographic hash function that is one-way and collision resistant.
2. They apply the hash function to each element of their own set.
3. They share all their hash values and compare which ones are equal. The elements corresponding to the common hash values form the intersection.

The security of this protocol relies on a large universe $\mathcal{U}$ and a sufficiently-close-to-ideal hash function. In particular, if $\mathcal{U}$ is too small, then both parties can just test all elements and compare to the other's hash values. Various improvements on this naive algorithm lessen the importance of these assumptions. Modern approaches include replacing the hash function with a public-key encryption, or using fully homomorphic encryption [4], or using oblivious transfer instead [8].

It is worth mentioning that while set intersection is a primitive that can be used in larger algorithms, it also has direct applications by itself. For example, it can be directly applied to networks of people, be it for finding common friends, for contact tracing, or for other scenarios.

## 3.6 Summation

**Problem 6.** Given $n$ elements $a_n$ of a field $F$, determine $\sum_n a_n$.

We present a protocol for this problem based on Shamir's secret sharing scheme [5]. The key observation is that Shamir's secret sharing scheme allows one to compute any linear combination of secrets.

1. The $n$-party agrees on $n$ public random element $X_n$.
2. Each party $P_i$ chooses a random polynomial $p_i$ of degree $n-1$ and constant term $a_i$.
3. Each party sends each other party $P_j$ their share $p_i(X_j)$
4. Each party adds up all the shares it receives from other parties, and sends this result $I_i$ to all other parties.
5. Now each party has the value of the polynomial $p = \sum_n p_n$ at all $X_n$, they can determine the constant term of $p$, which corresponds to $\sum_n a_n$

The security of this protocol relies on the security of Shamir's secret sharing scheme. Because for each party $P_i$, all other parties only get hold of $n-1$ shares of its secret value $a_n$, it is impossible for them to recover $a_n$ even if they conspire collectively. [9]

### 3.7 Applications to Bigger Problems

We selected the above list of primitives based on their fundamental nature as well as usage in other, larger problems. We have already seen some primitives applied others. For example, oblivious transfer was used to construct garbled circuits, garbled circuits were used for many of the simpler primitives, and the min primitive was used for set union. Here we provide a list of examples of larger applications of these primitives, along with their references for implementation details which are not the focus of this work.

- All pairs shortest distance in a graph [3]
- Single source shortest distance in a graph [3]
- Unlabelled graph construction from partial information [7]
- Decision tree inference using ID3 construction [5]

There are also various higher-level domain-specific MPC primitives which deviate in style from our collection. A prominent example of this is the vector dot product, and a couple implementations are provided in [1]. The dot product proves to be very useful in many fields, including data analysis, machine learning, and computational geometry. We provide a similar list for applications of the dot product.

- Determining whether a vector dominates another [1]
- Determining whether a point is in a polygon [1]
- Determining whether two polygons intersect [1]

## 4 Lattice Meet and Join

We note that many of the primitives presented in the prior section are special cases of a mathematical object known as *order lattices*. These lattices (not to be confused with group-theoretic lattices) generalize many useful structures such as sets, real numbers, topological spaces and program semantics. Therefore, there has been both theoretical and practical interest in studying lattices. In this section we present the mathematical facts about lattices following [10] and reframe our existing primitives in the framework of lattices.

**Definition 2.** A *partially ordered set (poset)* is a set $S$ with an ordering relation $\leq$ on some (not necessarily all) elements satisfying the following properties:

- $x \leq x$ for all $x \in S$.
- If $x \leq y$ and $y \leq x$, then $x = y$.
- If $x \leq y$ and $y \leq z$, then $x \leq z$.

Several of the structures we studied are posets. For example, the integers with their natural ordering form a poset (in fact a totally-ordered set). The power set (set of all subsets) of a given set $\mathcal{U}$ also form a poset, where for two subsets $x, y \in \mathcal{P}(S)$, we define $x \leq y$ to mean $x \subseteq y$. This is an example of a true partial order, since there exist sets $x, y \in \mathcal{P}(S)$ where neither $x \subseteq y$ nor $y \subseteq x$ is true; that is, they are incomparable elements under the subset ordering.

**Definition 3.** A *lattice* is a poset such that for every two elements, there exists a unique least upper bound and a unique greatest lower bound. These are called the *meet* and *join*, respectively.

There is also a more algebraic and axiomatic definition of lattices.

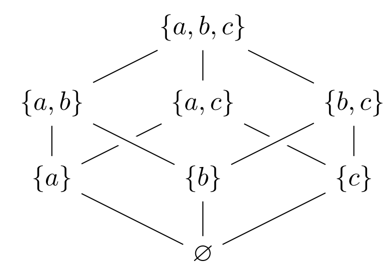
**Definition 4.** A *lattice* is a set $S$ equipped with two binary operations $\vee$ (*join*) and $\wedge$ (*meet*) that are commutative, associative, and satisfy the absorption laws

$$x \vee (x \wedge y) = x \qquad x \wedge (x \vee y) = x$$

for all $x, y \in S$.

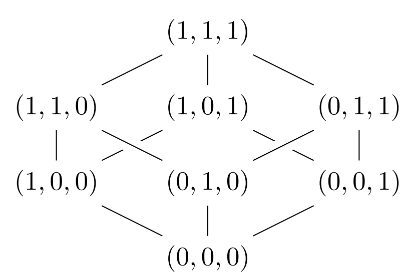
These two definitions are equivalent. Starting from poset definition, one can prove the properties in the axiomatic definition. Starting from the axiomatic definition, one can reconstruct the poset by defining the ordering relation as $a \leq b \iff a = a \wedge b$. We leave further mathematical details to [10].

Both of our aforementioned posets are also lattices. In fact, the lattice join and meet operations $\vee, \wedge$ on a lattice can be seen as a generalization of the set union and intersection operations $\cup, \cap$. The following Hasse diagram shows an example of a poset on the subsets of a 3-element set, with upward edges indicating the inclusion relation.




A number of the previously described primitives are special cases of the MPC problem for lattice meet and join.

- The minimum (maximum) problem is the lattice meet (join) problem on integers, equipped with the natural total order.
- The set union (intersection) problem is the lattice join (meet) problem on power set lattices.
- The bit OR (AND) problem is the lattice join (meet) problem on the Boolean lattice $\{0,1\}^n$.




Note that the Boolean lattice $\{0,1\}^n$ is in fact isomorphic to the lattice on $n$-element subsets, where a 1 bit in the $i$th position corresponds to including the $i$th element in the subset, and a 0 bit corresponds to not including the element.

Given all these connections, it is natural to pose the following new problem.

**Problem 7.** Given a lattice $L$ and two elements $a, b \in L$, determine $a \vee b$ (resp $a \wedge b$).

We can also ask the more general *n-MPC problem for lattice meet/join*: Given $n$ parties, with $i$-th party holding a private element $p_i$ in a public lattice $L$, compute the meet/join, or some combination thereof, of $p_1, p_2 \cdots p_n$ in a privacy-preserving manner.

This is a very general problem that encompasses nearly all of the aforementioned primitives. As with any MPC problem, it is hypothetically solvable via Yao's garbled circuits, though would not necessarily be efficient. Any improvement over the generic method would have ramifications trickle down to all specific cases of lattice meet/join, making this problem a worthwhile one to pose. At the same time though, its generality also causes its difficulty.

Another source of difficulty is in representing the lattice inputs, even for non-privacy-preserving algorithms. For small lattices, it may be reasonable to specify the entire lattice as a graph for its Hasse diagram, from which one may apply graph algorithms. However, general lattices of interest are much larger, in which case specifying the entire graph is impractical. Mathematiaclly, lattices oftentimes span infinite or continuous domains (which may get discretized for input to computers).

Because of this, in general it is more fruitful to specify the meet and join functions via the axiomatic definition as opposed to the poset definition. While the $n$-MPC problem for lattice meet/join is a very general problem, we observe that studying the 2-MPC case is unlikely to give any useful results. This is because the meet and join can be any two arbitrary computable functions satisfying the axioms. However, given two arbitrary computable functions, we cannot possibly do better than general 2-MPC protocols. For this case, the only improvements would come from applying the axioms to restrict what kinds of functions are possible, yet we've seen the wide generality of functions and lattice that are possible.

On the other hand, it's meaningful to ask given a protocol for 2-MPC lattice meet/join (either the general garbled-circuit protocol, or an efficient specialized protocol), can we derive a protocol for the corresponding $n$-MPC lattice meet/join? We expect it is possible to do better than the most general case because of the axiomatic properties lattices satisfy, particularly absorption for reducing combinations of 3 terms. If the answer to this problem is indeed positive, we can derive various efficient $n$-MPC protocols because efficient 2-MPC protocols are known for a wide range of lattice meet/join problems. Due to the complexity of $n$-MPC problems and the limited scope of this work, we do not give an answer but rather pose this as an open problem for the MPC research community.

# References


[1] Mikhail J. Atallah and Wenliang Du. Secure multi-party computational geometry. In Frank Dehne, Jörg-Rüdiger Sack, and Roberto Tamassia, editors, *Algorithms and Data Structures*, pages 165–179, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[2] Peter Bogetoft, Dan Lund Christensen, Ivan Damgard, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. Multiparty computation goes live. Cryptology ePrint Archive, Report 2008/068, 2008. `https://eprint.iacr.org/2008/068`.

[3] Justin Brickell and Vitaly Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. In Bimal Roy, editor, *Advances in Cryptology - ASIACRYPT 2005*, pages 236–252, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[4] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1243–1255, New York, NY, USA, 2017. Association for Computing Machinery.

[5] Fatih Emekçi, Ozgur D Sahin, Divyakant Agrawal, and Amr El Abbadi. Privacy preserving decision tree learning over multiple parties. *Data & Knowledge Engineering*, 63(2):348–361, 2007.

[6] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, June 1985.

[7] Varsha Bhat Kukkala, Jaspal Singh Saini, and S.R.S. Iyengar. Secure multiparty computation of a social network. Cryptology ePrint Archive, Report 2015/817, 2015. `https://eprint.iacr.org/2015/817`.

[8] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 515–530, Washington, D.C., August 2015. USENIX Association.

[9] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[10] R.P. Stanley and G.C. Rota. *Enumerative Combinatorics: Volume 1*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1997.

[11] A. C. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167, 1986.

[12] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, page 160–164, USA, 1982. IEEE Computer Society.