Secure Multi-party Computation: A Survey

A Comparison of Secure Multi-party Computation Protocols and other Techniques for Computing on Encrypted Data

Pedro Gomes Moreira[1]

Supervisor: Lilika Markatou[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Pedro Gomes Moreira
Final project course: CSE3000 Research Project
Thesis committee: Lilika Markatou, Tim Coopmans

Abstract

Secure multi-party computation (MPC) allows parties to compute on their secret inputs, without revealing them to each other. As an area of theoretical interest, many MPC protocol have been developed in the last four decades. They each present different characteristics and are classified under distinct categories depending on their generality, security assumptions, and functionality. More recently, MPC has also become an area of practical interest due to optimizations in performance of the protocols. In this paper, we compare MPC protocols and other techniques for computing on encrypted data, considering how their properties affect security, efficiency, usability, and functionality. We show that there is a trade-off between security and efficiency when different adversarial models are used, as well as a trade-off between efficiency and flexibility in specialized protocols.

1 Introduction

Consider a hospital that wants to discover if their patients have a high risk of having cancer by comparing their DNA with those of cancer patients [36]. With such sensitive information in hand, an ethical and legal problem arises of how these data can be used in a way that guarantees both privacy of the patients and the desired functionality. An ideal solution would be to perform computations on the data in a private manner, revealing only whether a certain patient has high risk of cancer or not, while revealing nothing else about the patient.

This scenario illustrates the idea of secure multi-party computation (MPC), in which parties want to compute a public function using each of their inputs, while preserving these inputs private to other parties [21; 36]. A classical example of MPC is known as Yao's Millionaires' Problem [54], which can be formulated thus: Two millionaires want to know who is richer. However, they do not want to reveal any additional information about their wealth. In this case, the function to be computed could be as follows:

$$f(x,y) = \begin{cases} 0 & x < y \\ 1 & otherwise \end{cases}$$

Where x and y represent the wealth of each millionaire. More generally, the problem to be solved is computing the function $f(x_1, x_2, ..., x_n)$ while ensuring both correctness (the output is according to the defined function) and privacy (nothing is revealed beyond the output) [37].

To perform secure computations, multiple protocols have been developed, which can be classified in different categories. *Two-party* (2PC) and *multi-party* (MPC) protocols differ in how many parties they support: two or multiple. *Generic* protocols can be used to compute any function, while *specialized* ones can only (efficiently) be applied to compute specific functions [21]. Protocols also differ in the adversarial model they follow: a *semi-honest* adversary, also called *passive*, or *honest-but-curious* [21] is an adversary controlling parties who follow the protocol faithfully, but attempt to gather information about another party's secret [11], which poses a threat to the privacy constraint. On the other hand, a *malicious* (or *active*) adversary controls parties who do not follow the protocol, but can execute malicious code [11], which compromises correctness as well as privacy. Some protocols offer resistance to a semi-honest adversary alone, while others apply techniques to also protect against a malicious adversary; furthermore, protocols differ in how many parties can be corrupted by an adversary.

In this paper, we provide a systematic comparison of different MPC protocols and, in addition, between MPC and other techniques. The comparison is in terms of security, efficiency, usability, and functionality. Sub-questions are answered according to each category of protocols described above. Generic and specialized protocols are compared, investigating how much performance improvement a specialized protocol can give compared to a generic one and in what circumstances. The impact of different adversarial models (semi-honest or active) in terms of security and efficiency is also treated. In addition, there is an exploration of how a semi-honest protocol can be converted into a malicious-secure one. Possible optimizations to the protocols and the extent to which they improve their efficiency are investigated. Lastly, MPC is compared with four other techniques for computation with encrypted data, namely: (fully) homomorphic encryption, oblivious RAM, structured encryption, and trusted execution environments.

The following structure is followed in this paper: Section 2 covers the methodology followed to conduct the literature survey. Section 3 gives the background knowledge required to understand the protocols, which are described later in Section 4. Section 5 considers these protocols in terms of security, efficiency, usability, and functionality. Section 6 compares MPC with other techniques for secure computation. Issues of ethics and research integrity are considered in Section 7. Finally, Section 8 summarizes the findings of the paper with tables and presents unanswered questions.

1.1 Related Work

Various papers have been written in the field of MPC. Lindell [36] gives an overview of MPC, along with the techniques most used to perform it, and mentions state-of-the-art solutions. As a more extensive and detailed view of the field, Evans et al. [21] wrote a book, from the fundamental techniques to the most recent developments. In another paper, Orsini [46] gives a survey focused on MPC techniques for active security with an honest majority of parties. Hastings et al. [27] present a SoK on MPC compilers, which can translate code to MPC protocols. A technical review on MPC tools in the context of IoT and data analytics is presented in a paper by Raeini and Nojoumian [22]: these tools build on the fundamental protocols that will be discussed in this paper. Finally, and more related to the goal of this paper, Perry et al. [47] present a tool that compares MPC protocols based on environmental features, assumptions, security, and efficiency. However, their paper does not present the comparison, but only the framework.

Compared to these previous works, our paper focuses on the fundamental MPC protocols and compares them at a theoretical level, without benchmarks. Practical applications and frameworks are considered in terms of which protocols they use and for which reason. This direction of research aims to fill a literature gap of a systematic comparison of MPC protocols.

2 Methodology

In this literature survey, three approaches were used to select papers: snowball sampling, sharing papers in the research group, and using a search database.

Snowball sampling was used to gather papers in all relevant topics of this research. From a first introductory paper [36] and a book [21], a backward referencing search was performed for each of the protocols and other relevant topics that are part of the research sub-questions. The protocols were selected for research based on a chapter in the books from Evans et al. [21].

Within our research team, we shared papers on each one of the four techniques mentioned in Section 1. Hence, all the papers on these techniques were suggested by peers, except a paper from Gordon et al. [26] which was gathered from the book from Evans et al. [53]. This approach was reasonable because each member of the team is more acquainted with his own technique and can thus provide more insightful material to the others.

Lastly, Scopus was used to collect more up-to-date literature and to obtain a clearer picture of state-of-the-art technologies. Forward referencing was also occasionally performed to reach more recent papers. More details on the queries used for literature selection on Scopus are found in Appendix A.1. For each paper gathered with these three approaches, the relevant sections were read and annotated for future usage.

3 Background

In this section, we explain two important techniques that are used by MPC protocols: oblivious transfer and Shamir's secret sharing.

3.1 Oblivious Transfer

Oblivious transfer (OT) was originally introduced by Rabin [50] as a way of transmitting information between two parties, where the sender does not know what the recipient received. This idea was generalized into a 1-out-of-2 OT [43], where the sender has two values, say a_0 and a_1, and the receiver has a bit $b \in \{0, 1\}$, and wants to learn a_b from the sender. Then, this primitive allows the receiver to learn a_b but have no knowledge of a_{1-b}, while the sender does not learn b. More generally, one can consider a 1-out-of-n OT, or k-out-of-n OT, which typically build upon the 1-out-of-2 OT.

This protocol is significant in the context of MPC because, as shown by Killian [30], it is sufficient to construct a secure function evaluation for any computable function. Moreover, as shown in Section 4, some protocols make use of OT as a cryptographic primitive.

3.2 Shamir's Secret Sharing

Sharmir's secret sharing was introduced by Shamir [52] as a way to divide a value D into n shares, so that with at least $k \leq n$ shares it is possible to retrieve the value of D, but with $k-1$ or less shares, one cannot learn any information about D. With the parameters defined above, this is referred to as a (k, n) or k-out-of-n [36] threshold scheme.

In order to achieve this, the secret sharing scheme makes use of polynomials and the fact that given k points in a two-dimensional plane, there only exists a single polynomial of degree $k-1$ that passes through all the points. For a k-out-of-n threshold scheme, the idea is to generate a polynomial $g(x)$ of degree $k-1$, along with n points $(x_1, y_1), ..., (x_n, y_n)$ such that $g(x_i) = y_i$ for each $i \in \{1, ..., n\}$. These points are then distributed to n parties, and the secret that they share is $g(0)$. This value can only be found if the polynomial is known, which can only be achieved with at least k points. With knowledge of $k-1$ or fewer points, it is impossible to know what the secret is, since every polynomial is equally likely [36]. This provides information-theoretic security to the scheme, a concept which is later discussed in Section 5.1.

4 Protocols

In this section, four fundamental MPC protocols are explained, following the structure presented by Evans et al. [21]: Yao's GC, GMW, BGW, and BMR. A description of each protocol follows, along with possible optimizations.

4.1 Yao's GC

Protocol Description

Yao's garbled circuits (GC) protocol was introduced by Andrew Yao [55] in 1986. A detailed explanation of Yao's GC (in addition to a rigorous proof of its security) is provided by Lindell and Pinkas [37]. Here, we also provide an explanation of the protocol based on their paper.

There are two parties involved: the constructor (also called sender or garbler) P_1 and the evaluator (receiver) P_2. The protocol follows these steps: garbling, encrypting, permuting, running, and decoding the output.

Garbling: The first step is to represent the functionality to be computed as a Boolean circuit, which is done by P_1. This circuit is garbled, which means that the values of the gate wires are hidden from both parties. Concretely, each wire i has wire labels k_i^0 corresponding to bit 0 and k_i^1 corresponding to bit 1. These labels will be used instead of values, so that the semantics of the circuit are unknown. For example, consider the truth table of an AND gate (Table 1). There are two input wires i and j, and the output wire k. Each 0 in column i is replaced by k_i^0, and each 1 by k_i^1 (and likewise for the other wires j and k). The resulting table is shown (Table 2).

Table 1: Truth table of the AND gate.

Wire i	Wire j	Output wire k
0	0	0
0	1	0
1	0	0
1	1	1

Table 2: Garbled truth table of the AND gate, based on [37].

Wire i	Wire j	Output wire k	Garbled table
k_i^0	k_j^0	k_k^0	$E_{k_i^0}(E_{k_j^0}(k_k^0))$
k_i^0	k_j^1	k_k^0	$E_{k_i^0}(E_{k_j^1}(k_k^0))$
k_i^1	k_j^0	k_k^0	$E_{k_i^1}(E_{k_j^0}(k_k^0))$
k_i^1	k_j^1	k_k^1	$E_{k_i^1}(E_{k_j^1}(k_k^1))$

Encrypting: The next step is to encrypt the output wire labels using the input wire labels as symmetric keys. For a certain output of the table, the labels of the corresponding two inputs are used. This is done so that, with two input keys, only a single output label can be decrypted. The result is shown in Table 2.

Permuting: Importantly, the rows of the garbled table must be permuted; otherwise, the evaluator, upon receiving it, would know that the output label of the first row corresponds to the labels 00, the second row to 01, etc. This process of garbling and permuting is repeated for each gate in the circuit. When this is finished, P_1 sends the garbled circuit to P_2, which includes the garbled tables of all gates.

Running: The evaluation of the circuit follows a topological ordering. For each gate, one wire is owned by each player. P_1 can simply send the wire label corresponding to his input to P_2. The other wire label, corresponding to P_2's input, is selected through 1-out-of-2 OT, where P_1 is the sender and P_2 the receiver. Here, P_2 selects one label out of two, without discovering the other one, and P_1 does not learn which option P_2 selected. With the two input wire labels, P_2 can evaluate the output label by decrypting the correct row of the garbled table received from P_1. Depending on the circuit, the output label may be the input label of another gate, in which case the process is repeated.

Decoding the output: Finally, once the final gates with no successor are evaluated, P_2 should find the real wire values corresponding to the output labels of the output gates. For this purpose, P_1 also sends a table that maps these final output labels with the real values. Hence, P_2 will be able to retrieve the output values and share them with P_1.

Optimizations

Here, we describe four optimizations: point-and-permute, FreeXOR, garbled row reduction, and half-gates.

One minor issue in the protocol described above is that P_2 has to potentially decrypt each one of the four rows from the garbled table with the two input label keys, and check if the resulting plaintext is gibberish, or a meaningful value. In the latter case, P_2 knows that this is the correct row to decrypt, and uses that output label. To solve this inconvenience, Beaver et al. [4] introduced **point-and-permute**. Its idea is that P_1 indicates to P_2 which row of the garbled table should be decrypted, by using the first bits of the keys as pointers to the garbled table [21].

FreeXOR is a technique developed by Kolesnikov and Schneider [32], where XOR gates are evaluated "for free", that is, in their own words, "without the use of the associated garbled tables and the corresponding hashing or symmetric key operations" [32, 2]. The technique adapts a construction from [31] to garbled circuits. The idea is to make the wire labels dependent on each other, so that the output label can be determined by XORing the input labels.

Another technique used in GCs is **garbled row reduction** (GRR), introduced by Naor et al. [44]. Here, the aim is to reduce the size of the garbled table, hence sending only three rows (GRR3) instead of four for each table. This is done by P_1 choosing an output label such that encrypting the row returns a null string. Then, P_1 only needs to send three non-null rows to P_2, since the latter can assume that there will always be a null row. Further developments in GRR were made by Pinkas et al. [48], who, using polynomial interpolation, constructed a row reduction that only required two ciphertexts (GRR2) sent per AND gate. However, this technique was not compatible with FreeXOR.

To solve this incompatibility, Zahur et al. [56] introduced **half-gates**, as a way to combine FreeXOR with GRR2. Their insight was to write AND gates as a combination of XOR gates, and half-gates, which are AND gates where one of the wire values is known by one party. By doing this, AND gates use only two ciphertexts, while XOR gates use none. The authors showed that for all circuits, the technique leads to a smaller circuit size than all previous methods.

4.2 GMW

Protocol Description

GMW is a protocol named after Goldreich, Micali, and Wigderson [24], who introduced it in 1987 as a way to solve the secure multi-party problem given a majority of honest parties.

The protocol has some differences compared to Yao's GC. First, it can naturally handle more than two players. Moreover, despite still converting the function to a circuit, it does not garble the circuit, but instead does secret sharing of the wire values. A third difference is that GMW can evaluate both Boolean and arithmetic circuits, since there is a correspondence between addition and XOR, and multiplication and AND [25].

An explanation of the protocol is given by Evans et al. [21] and Goldreich [25], which we partially follow in our presentation. We consider the protocol as a Boolean circuit evaluation between two parties for simplicity without loss of generality. The protocol has the following steps: sharing the inputs, evaluating the circuit, and recovering the output.

Sharing the inputs: Each party has a bit string $x \in \{0,1\}^n$, where n is the length of the input. Each party generates a random bit string $r \in \{0,1\}^n$ that serves as a mask to his input. Then, for each bit i of the input, the party sends r_i to the other party as a share of x_i, and keeps $x_i \oplus r_i$ as his own share. Note than XORing the shares gives the secret: $r_i \oplus (x_i \oplus r_i) = x_i$.

Evaluating the circuit: The two parties evaluate the circuit gate by gate, in topological order. For a gate with two input wires, each party $p \in \{1, 2\}$ holds a share s_i^p of each input wire i. An additive sharing scheme is used, where the wire value $w_i = s_i^1 \oplus s_i^2$. We consider three types of gates: NOT, XOR, and AND, which are functionally complete.

- NOT: can be evaluated locally by having one party p flip his share s_i^p of the single input wire i.
- XOR: Both parties can locally XOR their shares, and this will result in shares of the output, since $(s_i^1 \oplus s_j^1) \oplus (s_i^2 \oplus s_j^2) = (s_i^1 \oplus s_i^2) \oplus (s_j^1 \oplus s_j^2) = w_i \oplus w_j = w_k$, which is the desired output value.
- AND: Here, the parties have to interact and cannot compute the gate locally. What we desire are two output shares s_k^1 and s_k^2 such that $s_k^1 \oplus s_k^2 = w_k = w_i \wedge w_j = (s_i^1 \oplus s_i^2) \wedge (s_j^1 \oplus s_j^2)$. To retrieve this, the parties engage in a 1-out-of-4 OT where P_1 is the sender and P_2 the receiver. The idea is that P_1 considers all four possibilities of the two shares s_i^2 and s_j^2 of P_2, which can be 00, 01, 10, or 11. Then, P_1 generates all four possible outputs $(s_i^1 \oplus s_i^2) \wedge (s_j^1 \oplus s_j^2)$, and computes the XOR of each output with a random bit mask $r \in \{0, 1\}$, and these four results become his input in the OT. P_2 chooses one of them according to his two shares, so that P_1 does not discover which one P_2 selected. Then, this selection will be P_2's share of the output, while r will be P_1's share.

Recovering the output: To retrieve the output, each party sends to the other the shares of the output wires of the circuit.

Optimizations

Here, we describe one possible optimization to GMW: offline and online phases. An offline phase, or pre-processing phase, refers to the protocol execution when the parties' inputs are not yet known, whereas in the online phase the inputs are known [21]. As also noted by Evans et al. [21], the purpose of this separation is to move as much processing to the offline phase, so that the online phase can run efficiently. In the pre-processing phase of GMW, it is possible to pre-compute OTs for evaluating AND gates or using Beaver's multiplication triples for the same purpose: Schneider and Zohner [51] show that the latter approach leads to better results. The idea of this technique is to generate a triple of secret shared values, where one value is the product of the other two, which allows for evaluating AND gates only with local computations in the online phase [21].

4.3 BGW

Protocol Description

BGW is a protocol published by Ben-Or, Goldwasser, and Wigderson [6] in 1988. It was developed simultaneously with a similar protocol by Chaum, Crépau, and Damgård (CCD) [15]. The protocol is designed in such a way that it is secure against a semi-honest adversary with an honest majority, and secure against a malicious adversary with $t < n/3$ corrupted parties.

The protocol has some similarities with GMW, namely that it can handle more than two parties, and uses secret sharing on the wire values. The main differences with GMW are that it uses Shamir's secret sharing instead of an additive sharing, and that it can only be used to evaluate arithmetic circuits.

We provide an explanation based on the work from Evans et al. [21]. The protocol has the same steps as GMW, which are the following: sharing the inputs, evaluating the circuit, and retrieving the outputs.

Sharing the inputs: Each party holding an input wire value s generates a polynomial $p(x)$ of degree t, such that $p(0) = s$, where $t < n/2$ is the number of corrupted parties in the semi-honest model. The player then distributes shares of s to all other parties, where each share is a point in a polynomial $p(x)$.

Evaluating the circuit: Three types of arithmetic gates are considered: multiplication of polynomial by constant, addition of two polynomials, and multiplication of two polynomials.

- Multiplication by constant: Each party can locally compute a share of the output by simply multiplying the share of the input with the known constant. This holds because multiplying each y-value of the points of $f(x)$ by c results in points of the polynomial $h(x) = cf(x)$. Hence, the points of this polynomial are valid shares of the secret $h(0) = cf(0)$.
- Addition: Likewise, given that each party has a share of each of the two polynomials, the player can locally add the two shares, and will have a share of the output. Adding the points of the two polynomials $f(x)$ and $g(x)$ will give points of the addition $h(x) = f(x) + g(x)$, which constitute valid shares of the secret $h(0) = f(0) + g(0)$.
- Multiplication of polynomials: The product of the polynomials $h(x) = f(x)g(x)$ has degree $2t$, which means that multiplying the points will result in shares of a $2t$-out-of-n threshold scheme. This presents a problem, because subsequent multiplications can raise the degree to be above n, which will result in insufficient points to reconstruct the polynomial, and hence to find the secret. Thus, a degree reduction procedure is necessary, which involves interaction between the parties.

Retrieving the outputs: Finally, after all gates are evaluated, the parties will each have a share of the output. The parties can all broadcast their shares, so that they can reconstruct the output value.

Optimizations

In BGW, it is also possible to user Beaver's multiplication triples, which leads to evaluation of one multiplicative gate with two openings (of the triples) and a local computation [21].

4.4 BMR

Protocol Description

BMR is a protocol developed by Beaver, Micali, and Rogaway [4], which was published in 1990. Their original paper and the book from Evans et al. [21] provide a detailed explanation of the protocol. Here, we follow a more simplified and

abstract overview, in line with a paper from Lindell et al. [39] and one from Ben-David et al. [5].

Compared to other protocols, BMR is an extension of Yao's GC to the multi-party case, and hence it distances itself from secret sharing protocols such as GMW and BGW. A further difference is that it only handles Boolean circuits. On the other hand, like BGW, it is also secure against a semi-honest adversary with an honest majority of parties.

The idea behind the protocol is to perform a distributed computation, where all parties are involved in both garbling and evaluating, contrary to Yao's GC where each party has one role. The protocol has two phases: an offline phase where the parties create the garbled circuit and the garbled inputs, and an online phase where each party evaluates the garbled circuit individually.

Offline phase: Each party p, for each wire i of the circuit, will generate two so called *seeds*: $s^0_{i,p}$ for value 0, and $s^1_{i,p}$ for value 1. The concatenation of all the seeds of the players gives the *superseed* $s^0_i = s^0_{i,1}||...||s^0_{i,n}$ for n parties (likewise for s^1_i). In the terminology of Evan et al, the superseed is the wire label (as in Yao's GC), and the seeds are the sublabels of which the label is composed. These superseeds must be randomized, so that the player cannot know that the first superseed received corresponds to 0 and the second to 1. This randomization is done using a *flip bit*, which is XORed to the real value (0 or 1), to produce and *external value*. Hence, a party may know that a superseed is associated with a certain external value, but cannot gain any information on the real value.

For each gate, the garbled table is built by encrypting the output superseed using the two corresponding input superseeds, as described for Yao's GC. This results in the essentially the same table as Table 2.

Online phase: Each player who is assigned a input wire broadcasts the corresponding superseed to all players. All players, knowing the garbled tables, can evaluate each gate with the superseeds and retrieve the output superseed. Again, as in Yao's GC, this final superseed can be translated to the real value.

Optimizations

Built-in BMR is the division between offline and online phases. The online phase can be made particularly fast by using pre-processing with techniques derived from other protocols, as shown by Lindell et al. [39].

5 Characteristics

In this section, we consider the MPC protocols of Section 4 in terms of security, efficiency, usability, and functionality.

5.1 Security

In this section, we give a comparison of protocols in terms of active security, number of colluding parties, and information-theoretical security.

What distinguishes the semi-honest from the malicious adversarial model is the capabilities of the corrupt parties. In the semi-honest model, corrupt parties must follow the protocol, but they keep track of all data they receive and can run computations on these data, in order to discover more information about the other parties' inputs, whereas in the malicious model, corrupt parties can arbitrarily deviate from the protocol [37; 38]. To give a concrete example of this difference, consider Yao's GC [55]. If followed faithfully, the protocol reveals no information on the wire values, except the output; hence, it provides semi-honest security (a rigorous proof of this fact was presented by Lindell and Pinkas [37]). However, a malicious party P_1 can garble a circuit that computes a function different from the one P_2 agreed to compute. For example, in the Yao's Millionaire's Problem, P_1 could construct a circuit that computes $f(x, y) = y$, meaning that the output of the circuit is the input (in this case, the wealth) of P_2.

To protect against such a threat, some techniques exist for adapting a protocol to be secure against malicious adversaries: we consider cut-and-choose and the GMW compiler. **Cut-and-choose** is a technique that has been applied in Yao's GC for two parties by Lindell and Pinkas [38]. It aims to defend against a corrupt circuit sent by P_1 from being evaluated. The basic idea is that P_1 sends copies of the garbled circuit to P_2, who selects some of them to be "opened" by P_1, and P_2 then verifies that these have been constructed correctly. Later, P_2 evaluates the unchecked GCs and considers the majority value as the final output. This is done to prevent an incorrect circuit from being evaluated: it is very unlikely that most of the unverified circuits will be corrupt. The **GMW compiler** is a technique where parties produce so-called zero knowledge (ZK) proofs to show their honesty [38]. The compiler takes a semi-honest protocol p and returns an actively secure one with the same functionality, by proving in zero-knowledge that every message is a result of running p honestly, where zero-knowledge refers to the lack of knowledge of the private inputs [21]. As noted by Ben-David et al. [5] and Lindell and Pinkas [38], this approach is costly, since it adds ZK proofs for every step of the protocol. However, the GMW compiler is a generic way to make any protocol actively secure, not only those based on GC.

In both the semi-honest and malicious models, protocols often present a limit on the number t of colluding corrupted parties, which are in control of the adversary. This number is commonly considered as either an honest majority ($t < n/2$) or dishonest majority ($n/2 \leq t < n$). Protocols that use secret sharing, such as GMW [25], BGW [6], and CCD [15], as previously mentioned in Section 4, base their security on an honest majority of parties. This is necessary, because if a dishonest majority has shares to the secret in a (t, n) threshold scheme, it will be able to discover the secret. On the other hand, garbling protocols such as Yao's GC [55] and BMR [4] achieve security against a dishonest majority. In the case of Yao's GC, this is the only meaningful adversarial scenario, because an honest majority in 2PC implies that both parties are honest, which is a trivial case.

Information-theoretical security differs from computational security in that the latter relies on computational infeasibility of breaking a problem, while the former is impossible to break even by an adversary with unlimited computational power [42]. A protocol that assumes an honest major-

ity of parties achieves information-theoretical security [39], whereas a dishonest majority implies a computational setting [17]. However, with a dishonest majority it is still possible to have an information-theoretically secure online phase, which is the case of SPDZ [19; 46]. Passively secure linear secret sharing (LSS) protocols typically offer information-theoretic security, whereas 2PC schemes only offer computational security [3]: two examples of the former are BGW and CCD [36; 27]. In particular, BGW uses Shamir's secret sharing (cf. Section 3.2), which, given a number of shares lower than the threshold, provides no information on the secret.

5.2 Efficiency

In this section, we consider the efficiency of the protocols under two aspects: round complexity and circuit representation.

Round complexity refers to how many rounds of communications are necessary between the parties to complete the secure computation. A major difference exists between GC and LSS based protocols, namely, that GC has a constant number of communication rounds, whereas LSS has a number that is linear in the (multiplicative [27]) depth of the circuit [3]. However, GC has a larger bandwidth of communication (the entire circuit must be sent) [3], whereas LSS has low bandwidth [46]. Hence, there exists a trade-off between bandwidth and latency [21], where GC can perform better under high latency [31], while LSS strongly depends on low-depth circuits and low latency to perform well [10]. Due to their low bandwidth, protocols based on secret sharing achieve high throughput [33]. A concrete comparison of Yao's GC and GMW was performed by Schneider and Zohner [51]. They mention how Yao's GC was believed to be more efficient than GMW precisely because of its constant communication rounds, besides fewer OT operations. The authors show that, with optimizations, GMW is practical in semi-honest 2PC, with some advantages over Yao against an active adversary.

Another area of comparison is between arithmetic and Boolean circuits. The former are better to perform addition and multiplication operations, whereas the latter are more suited for comparison and Boolean operations and less efficient for arithmetic operations [48]. A more recent approach is to combine both types in so-called hybrid protocols to achieve optimal efficiency [27]: three concrete examples of this approach are Tetrad [33], Silph [16], and HyCC [10]. Nevertheless, switching between circuit representations is not without costs: conversion protocols are used to translate between circuit representations, which, though under continuous optimization, still contribute to the overall runtime. In the case of Silph [16], the authors investigated the number of conversions as an optimization problem.

5.3 Usability

In this section, we consider MPC protocols used in the context of frameworks and real-world applications.

The first MPC framework created was Fairplay [41]. It is a system designed for two-party computation, which compiles high-level code defined in a secure function definition language (SFDL) into a Yao's GC to be evaluated. The aim of the authors was to translate theoretical secure function evaluation (SFE) into a practical application. Fairplay can handle both semi-honest and malicious adversarial models, but with a difference in performance. FairplayMP [5] is an extension of Fairplay supporting more than two parties. The authors chose BMR as the base protocol, because it supports multiple parties and has constant communication rounds independently of the function, given that the communication rounds were a major bottleneck. These two frameworks, along with the above mentioned HyCC and Silph, consist of compilers, designed to aid with the creation of circuits, which can be a cumbersome and error-prone process [10]. They target users without technical knowledge on cryptography and circuits, so that secure computation becomes more accessible [10].

In its beginning, MPC was an area of theoretical interest to researchers, but as protocols were improved and optimized, MPC became an area of practical interest, with deployment in real-world cases [45]. The first large-scale practical application of MPC was a Danish sugar beet auction [9]. Previously, the importance of secure computation for auctions had been explained by Naor et al. [44]. The authors of [9] used a generic protocol based on Shamir's secret sharing, with three parties (servers) that computed on the encrypted data, along with a large number of input parties (clients). The protocol is semi-honest secure for servers, and actively secure for clients. The first use of MPC over a WAN (namely, the internet) was performed by Bogdanov et al. [8] in the context of financial data analysis . A similar approach was used with three servers, called data miners, and many clients. The authors deemed a generic MPC framework (namely, Sharemind [1]) beneficial, since new reports (inputs from clients) can be added with relative ease. In another application, Damgård et al. [20] used MPC to benchmark private data that were shared between a bank and a consultancy house. These two parties performed the computation, whereas the clients only sent and received data. A specialized protocol (a linear program solver) based on SPDZ was used for higher efficiency. Due to the high round-complexity of SPDZ, the authors recommend having the two servers in the same data center to minimize the communication overhead; however, doing so facilitates a malicious cloud service (holding both servers) to reconstruct the private data. Lastly, Lapets et al. [34] report on a usage of MPC in investigation of wage inequality. The collection of sensitive information (wages) raises ethical, legal, and institutional concerns; hence, MPC was deemed an appropriate solution to the problem. The authors desired a protocol with low resource consumption, usability and accessibility. In particular, participants lacking computational resources and technical cryptographic knowledge should be able to participate. To satisfy these requirements, they used an asymmetric client-server model where the servers alone perform computations, but not clients, who have more limited computational power. Moreover, the tool was deployed on the web for increased accessibility. They concluded that a framework for secure data analysis using MPC is promising for public research studies.

[1] https://sharemind.cyber.ee/

5.4 Functionality

Under this section, we consider the functionality of protocols, first defining generic and specialized protocols, and then comparing the two types in the case of private set intersection.

Generic and specialized protocols differ in the functions that they can compute: while generic protocols can compute any function, specialized ones are built in such a way that they achieve secure computation of a particular function or in a particular problem [21]. The protocols presented in Section 4 are all generic: they achieve this generality by converting an arbitrary function into a circuit, and evaluating that circuit privately. On the other hand, specialized or custom protocols can only solve a particular problem, and are often built on the assumption that generic protocols are less efficient, an assumption which was questioned in the work of Huang et al. [28].

One particular problem that MPC protocols can solve is computing the private set intersection (PSI). In this problem, two parties want to discover the intersection of private sets that they own, without disclosing any value that is not in the intersection [36]. While generic protocols can be used to solve this problem, specialized protocols have also been developed to solve this problem in particular. One such protocol is explained by Lindell [36]: the idea is that the two parties hash their values and compare their hashes, outputting the values whose hashes are present in both sets. In another instance, Ion et al. [29] present a specialized protocol to compute the private intersection-sum-with-cardinality, which is a PSI where one set has integer values associated with elements, and one wants to obtain the sum of these values of elements in the intersection, besides the cardinality (size) of the intersection. As noted by Pinkas et al. [49], while the most efficient specialized PSI protocols are faster than generic protocol computing PSI by two orders of magnitude, the former present the downside that computing another function requires changing the entire protocol, which might be difficult or even impossible; likewise, hardening the protocol against malicious adversaries may not be practically possible [21]. On the other hand, changing a circuit of a generic protocol is easier. Hence, there can be a trade-off between efficiency and flexibility depending on how much functionality a protocol is able to compute.

6 Other Techniques

In this section, we describe four other techniques for computing on encrypted data and how they compare to MPC: fully homomorphic encryption, oblivious RAM, structured encryption and trusted execution environments. In each subsection, the description of the technique is followed by a comparison with MPC in terms of its context, security, efficiency, usability, and functionality.

6.1 Fully Homomorphic Encryption

A homomorphic encryption (HE) scheme is a cryptographic primitive in which evaluating a certain circuit on plaintexts gives the same result as decrypting the outcome of evaluating the same circuit with the corresponding ciphertexts [23]. In other words, considering a function f instead of a circuit, $f(x_1, ...x_n) = Dec(f(Enc(x_1), ..., Enc(x_2))$, where Enc and Dec indicate encryption and decryption, respectively. Encrypting both sides of the equation, $Enc(f(x_1, ..., x_n)) = f(Enc(x_1), ..., Enc(x_n))$, which is an alternative definition: the encryption of the function taking plaintexts is equal to the function taking the corresponding ciphertexts. As an example, in an additive HE scheme, we have $Enc(x_1 + ... + x_n) = Enc(x_1) + ... + Enc(x_n)$.

Compared to MPC, FHE presents a similar **context**: parties computing arbitrarily on encrypted data [23]. FHE can be used as a primitive for MPC: given a FHE encryption scheme on the shares of the inputs, the parties can perform MPC by locally computing on the shares [19]. The **security** models of FHE are different than those of MPC: the former provides indistinguishability under chosen plaintext attack (IND-CPA), with some schemes providing indistinguishability under non-adaptive chosen ciphertext attack (IND-CCA1) [12]. Moreover, due to the public-key cryptography involved, FHE cannot provide information-theoretical security, which some MPC protocols can provide. In terms of **efficiency**, as observed by Archer et al. [3], HE is very communicationally efficient, only requiring communication to send inputs and to retrieve outputs. Compared to MPC, FHE requires less interaction, but this comes at the expense of computational efficiency, which is worse in FHE [16]. In the comparison of Archer et al. [3], this inefficiency is also visible in the fact that FHE solutions are not marked as market-ready (like GC and LSS), but as an academic prototype. In terms of **usability**, FHE, albeit theoretically appealing, is not competitive in practice with more traditional MPC approaches [19]. Its usage within MPC, HE comes with limitation, for example: partly HE (such as BGW [6] with additive homomorphism of Shamir's secret sharing), semi-homomorphism with relaxed requirements (as in BDOZ [7]), or somewhat HE with small circuits (as in SPDZ [19]). Finally, the **functionality** of FHE is the most generic, since, as already mentioned, it can be used to arbitrarily compute on encrypted data, which entails any possible function.

6.2 Oblivious RAM

Oblivious RAM (ORAM) is a technique designed for security in the context of outsourced data storage, where a client stores data on a remote server [53]. This server is untrusted, and therefore the client would like to hide the data from it. One option would be to encrypt the data stored: this is not sufficient, because an adversary can infer information about the data by looking at the access patterns, i.e., the blocks accessed by the client, and in which order they are accessed. To hide these access patterns, ORAM shuffles and re-encrypts data as they are accessed by the client. An ORAM scheme is secure if the requests are performed correctly and if the access patterns of two sequences of requests are computationally indistinguishable by anyone except the client.

Compared to MPC, ORAM presents a more specific **context**, namely that of outsourced storage between one trusted client and one untrusted server, whereas MPC considers the more general case of any number of parties. A further difference is that ORAM is concerned with storage and use of data rather than computation, which is the focus of MPC. In terms

of **security**, Gordon et al. [26] note that ORAM guarantees only one-sided security, where the client is trusted and the server is untrusted, with no security guarantees for the server against the client. They provide a solution by using a 2PC protocol to compute next ORAM instructions. On a similar note, ORAM provides security against a semi-honest server, but some constructions using server-side computation can also achieve malicious security [2]. In terms of **efficiency**, as also noted by Gordon et al. [26], a client performing a query for an item stored in a server requires at least linear complexity using a generic MPC approach, whereas ORAM constructions can achieve sublinear complexity. The authors provide a secure 2PC protocol in a client-server model using ORAM in order to achieve sublinear amortized complexity. Regarding **usability**, ORAM has been used in the context of MPC to design secure computation frameworks. For instance, Liu et al. [40] present a tool called ObliVM, which allows non-specialist users to compile code into secure MPC protocols, assuming a two-party semi-honest environment. A requirement of this process is memory-trace obliviousness, that is, hiding access patterns. One way to achieve this is by requiring that all data in the code be stored and accessed via ORAM: this is a generic and simple approach, but not always the most efficient. Other approaches are specific for certain algorithms and can outperform ORAM. The creators of ObliVM present a middle approach, attempting to first use the specialized solutions and falling back to ORAM when this fails. Lastly, in terms of **functionality**, ORAM focuses on hiding access patterns of read and write requests. As such, it does not consider computing arbitrary functionality, as MPC does.

6.3 Structured Encryption

Structured Encryption (StE) is a technique that allows private queries on encrypted data [14]. The context is of cloud storage, similar to that of ORAM: a client wants to store structured data on an untrusted server. Using encryption is not a desired solution, because the data would lose their structure and would thus not be able to be queried efficiently. StE presents itself as a solution that encrypts data in such a way that they can still be queried, but without revealing any information about the query or data. A more specific notion is searchable symmetric encryption (SSE), which can be seen as StE for a private keyword search over encrypted document collections, whereas StE can be applied to arbitrarily-structured data. SSE aims at scalability with large sizes of data: the techniques indeed achieve asymptotic efficiency, but the practical performance is doubtful, due to factors such as I/O latency [13].

Compared to MPC, the **context** of StE is more specific, as already noted for ORAM. The **functionality** of StE is therefore limited to performing queries on encrypted data, and does not extend to arbitrary functionality like MPC. In terms of **security** and **efficiency**, compared to other techniques such as HE and MPC , Cash et al. [13] note that SSE aims at practical efficiency, whereas the former techniques are highly secure but not practically efficient. On a similar note, Chase and Kamara [14] observe that, even though ORAM, 2PC, and FHE could be used to achieve the functionality of StE, it is preferable to seek non-interactive solutions, with at worst linear complexity in the length of the data. One visible example of such a difference between the techniques is the leakages that are present in StE schemes, such as of identifiers of documents that match a query, and the knowledge of repeated queries, as noted by Cash et al. [13]. Here, the authors also mention that that information could be hidden with the use of private information retrieval or ORAM, but it is not clear that ORAM is a competitive solution. In this trade-off between security and efficiency, StE focuses on practical efficiency and allows for some insecurities (even in a semi-honest model) for this purpose. Regarding **usability**, an interesting usage of StE to develop a MPC protocol was shown by Agarwal et al. [1] in the context of the already familiar PSI problem. Here, the problem considered was of *updatable* PSI, where the two sets grow or shrink over time, which happens for instance in online advertisement applications. The aim of the authors was to achieve a PSI protocol that supported arbitrary insertions and deletions in constant round of communications and sublinear complexity of computation. A solution was developed based on dynamic StE, which allows for such updates. A difficulty encountered by the authors was to limit the leakage of the underlying StE scheme to a defined minimum, which in this case they defined as the sizes of the update sets, whereas nothing else is revealed to any party. For client-side queries, a variation of ORAM was used, which leaks query equality (if the same query was performed multiple times), unlike traditional ORAM.

6.4 Trusted Execution Environments

Trusted execution environments (TEEs) are a technique used for secure outsourced computation [35]. Here, the context resembles that of ORAM: a trusted client and an untrusted provider. However, in this case, the focus is on outsourced computation rather than storage, and the provider is a cloud service provider (CSP) which grants computing resources to the client. Traditional cloud models require full trust in CSPs, so that they can control execution instances. This is an impediment to the larger usage of outsourced computation with sensitive data. Server-side TEEs present a solution to this problem by enabling confidential computation to protect sensitive workloads. However, there is still trust involved: namely, there is a trusted computing base (TCB) with all hardware and software components that are in the security foundation. Moreover, the hardware vendor is trusted by the client, which means that TEEs only remove the intermediate trust link between the client and the CSP.

Compared to MPC, TEEs have a more specific **context** of outsourced computation, where a trusted client and untrusted server are involved. On the other hand, the **functionality** of TEEs extends to any computation, and in this sense it is closer to MPC than ORAM and StE. To consider security, efficiency, and usability, we refer to a paper by Choid and Butler [18] that presents a survey of MPC and TEEs techniques that could be combined for increased performance. Related to **usability**, they observe that MPC is not yet practical for most applications where real-time performance is necessary, especially techniques based on FHE and LSS, whereas TEEs are on the rise. This is connected to **efficiency**: TEEs may present an

opportunity to offload costs of MPC, since the former have much faster execution without being tied to complex cryptography. However, they note that combining the two is not trivial: notably, their **security** assumptions and trust models differ. TEEs require additional security assumptions and a TCB, whereas in cryptography (MPC), the assumptions are simpler. In addition, care must be taken to avoid secret leakage when communicating between trusted and untrusted components of TEEs. The authors also point that defeating malicious adversaries could be done through remote attestation without cut-and-choose, but this would give the hardware vendor control over the attestation, meaning that it could perform a man-in-the-middle attack. Despite these differences and challenges, they point to preliminary works showing positive results in using TEEs for MPC.

7 Responsible Research

In this section, we consider issues related to bias, research integrity, reproducibility and replicability, and beyond the project.

Bias is present in every research, albeit slightly, in the way that researchers select papers and the extent to which they faithfully describe the information from the papers. In this literature survey, we attempted to use relevant and trustworthy sources that provide the necessary information to answer the research question. Hence, not all aspects of the field are considered. In particular, we focused on a theoretical discussion of the fundamental MPC protocols, rather than a benchmark of the newest developments in MPC frameworks. One noteworthy thing is that many papers used in this review were authored by the same individuals, who seem to be recurrent authors in the field of MPC. This might indicate that other researchers were left out of our paper selection, and their work not sufficiently represented.

The issue of bias also connects to research integrity. The sources used in this review were acknowledged, including inspirations used to explain the protocols. We also attempted to faithfully transmit the information present in the papers, not excluding any relevant information. This paper had no usage of generative AI besides suggestions by Writefull, which is an AI tool integrated in Overleaf. Licenses were not applicable, since we did not work with code.

Reproducibility and replicability ensure that other researchers can achieve the same conclusions presented in this paper, by using the same data (reproducibility) or different data (replicability). We believe that from the same papers selected for this review, the same conclusions can be reached. Furthermore, using the information given in Section 2, from a different set of initial papers, one could likewise reach the same conclusions, since they are agreed-upon ideas repeated in many papers. One potential issue of the methodology chosen for this paper is that papers were selected often arbitrarily: the choice of papers was not conducted according to clear rules, but by considering the relevance of one paper as perceived by us.

Lastly, there are issues to consider beyond the scope of the project. Namely, how could the conclusions presented here be used by readers? Having a systematic comparison of MPC protocols has beneficial aspects, for instance, helping developers to know which protocol or technique to use in which circumstances. Nevertheless, it will also benefit malicious users, who can easily learn which protocols are insecure and why; for example, which protocols are only semi-honest secure, or which protocols only provide computational security based on hardness assumptions. These considerations relate to the more fundamental question of whether public knowledge of vulnerabilities is beneficial or detrimental to security. While we do not attempt to fully answer this question, we follow Kerckhoff's principle that security should not be based on secrecy of cryptographic protocols, but rather of keys.

8 Conclusion

In this paper, we reviewed fundamental MPC protocols and compared them in terms of security, efficiency, usability and functionality. Moreover, the differences of the protocols were evaluated according to their categories. MPC was also compared to four other techniques for computing on encrypted data, considering their differences, similarities, and how they can be combined.

A number of conclusions could be gathered from this review. Regarding specialized protocols, they can outperform generic ones for certain tasks, but they are less flexible to extend and adapt, for example to compute (even slightly) different functions. Some specialized protocols were used in real-world applications, while in other cases, generic protocols were used, but all of the latter with a client-server model. Regarding adversarial model, a semi-honest model is weaker, but more efficient than a malicious one. Related to this fact, semi-honest protocols can be converted into maliciously secure versions of them with techniques such as cut-and-choose, but these results in extra costs on the complexity of the protocol. Regarding efficiency, some protocols can be optimized using protocol-specific techniques, and these have contributed to making MPC usable in practice. A comparison between MPC protocols is shown in Table 3.

Lastly, Tables 4 and 5 present a comparison of MPC with the four other techniques for computing on encrypted data. MPC stands on the side of generality and guaranteed security, whereas other techniques are more practically efficient. Moreover, MPC is quite flexible and can involve more choices than other protocols: the thread model can differ, as well as the efficiency, the generality of functions to be computed, the number of parties, etc. Part of these choices were surveyed in this paper.

For future work, further MPC protocols could be included in the comparison, such as BDOZ [7], SPDZ [19], etc. Moreover, one could consider more fine-grained security models including protecting against covert adversaries (who lie in between active and passive), adversarial mobility (static, adaptive or mobile), as well as fairness and guaranteed output delivery [47]. Efficiency could also be more thoroughly considered in terms of asymptotic vs. concrete efficiency, and protocols that focus more on one rather than the other.

A Appendix

A.1 Literature Selection

- **Query:** "MPC protocol*" AND compar*
- Filters:
  - Time frame: between 2018 and 2025
  - Subject area: computer science
  - Document type: conference paper, article
  - Language: English
  - Publication stage: final

A.2 Comparison Tables

References

[1] Archita Agarwal, David Cash, Marilyn George, Seny Kamara, Tarik Moataz, and Jaspal Singh. Updatable Private Set Intersection from Structured Encryption, 2024. Published: Cryptology ePrint Archive, Paper 2024/1183.

[2] Daniel Apon, Jonathan Katz, Elaine Shi, and Aishwarya Thiruvengadam. Verifiable Oblivious Storage, 2014. Published: Cryptology ePrint Archive, Paper 2014/153.

[3] David W. Archer, Dan Bogdanov, Benny Pinkas, and Pille Pullonen. Maturity and Performance of Programmable Secure Computation, 2015. Published: Cryptology ePrint Archive, Paper 2015/1039.

[4] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, STOC '90, pages 503–513, New York, NY, USA, 1990. Association for Computing Machinery. event-place: Baltimore, Maryland, USA.

[5] Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: a system for secure multi-party computation. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, pages 257–266, New York, NY, USA, 2008. Association for Computing Machinery. event-place: Alexandria, Virginia, USA.

[6] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 1–10, New York, NY, USA, 1988. Association for Computing Machinery. event-place: Chicago, Illinois, USA.

[7] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic Encryption and Multiparty Computation. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 169–188, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[8] Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis, 2011. Published: Cryptology ePrint Archive, Paper 2011/662.

[9] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. Secure Multiparty Computation Goes Live. In Roger Dingledine and Philippe Golle, editors, *Financial Cryptography and Data Security*, pages 325–343, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[10] N. Büscher, D. Demmler, S. Katzenbeisser, D. Kretzmer, and T. Schneider. HYCC: Compilation of hybrid protocols for practical secure computation. pages 847–861, 2018.

[11] Ran Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, January 2000.

[12] Ran Canetti, Srinivasan Raghuraman, Silas Richelson, and Vinod Vaikuntanathan. Chosen-Ciphertext Secure Fully Homomorphic Encryption. In *Proceedings, Part II, of the 20th IACR International Conference on Public-Key Cryptography — PKC 2017 - Volume 10175*, pages 213–240, Berlin, Heidelberg, 2017. Springer-Verlag.

[13] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation, 2014. Published: Cryptology ePrint Archive, Paper 2014/853.

[14] Melissa Chase and Seny Kamara. Structured Encryption and Controlled Disclosure. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, pages 577–594, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[15] David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 11–19, New York, NY, USA, 1988. Association for Computing Machinery. event-place: Chicago, Illinois, USA.

[16] E. Chen, J. Zhu, A. Ozdemir, R.S. Wahby, F. Brown, and W. Zheng. Silph: A Framework for Scalable and Accurate Generation of Hybrid MPC Protocols. volume 2023-May, pages 848–863, 2023.

[17] Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. Fast Large-Scale Honest-Majority MPC for Malicious Adversaries, 2018. Published: Cryptology ePrint Archive, Paper 2018/570.

[18] J.I. Choi and K.R.B. Butler. Secure Multiparty Computation and Trusted Hardware: Examining Adoption Challenges and Opportunities. *Security and Communication Networks*, 2019, 2019.

[19] I. Damgard, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption, 2011. Published: Cryptology ePrint Archive, Paper 2011/535.

Table 3: Comparison of MPC protocols, inspired by table in [27]. The traits considered fit under the four characteristics discussed in Section 5. A filled circle (black) indicates affirmative, and an unfilled circle (white) indicates negative. A half-filled circle under maliciously secure indicates that the protocol can be made malicious-secure with extensions (like Yao's GC) or in specific cases (like BGW with no more than one third of corrupted parties). The labels under practical usage were decided based on the practical applications discussed in Sections 5.3 and 5.4.

Protocol	Parties supported	Maliciously secure	Dishonest majority	Information-theoretically secure	Round complexity	Circuit type	Practical usage	Generic	Specialized
Yao's GC [55]	2	◐	●	○	Constant	Boolean	Large	●	○
GMW [24]	≥ 2	◐	○	○	Linear	Boolean/arithmetic	Medium	●	○
BGW [6]	≥ 2	◐	○	●	Linear	Arithmetic	Medium	●	○
CCD [15]	≥ 2	◐	○	●	Linear	Arithmetic	Little	●	○
BMR [4]	≥ 2	◐	●	○	Constant	Boolean	Medium	●	○
Specialized PSI [36]	2	○	●	○	Constant	N/A	Medium	○	●

Table 4: Functionality and Usability Comparison of Privacy-Enhancing Techniques

Technique	Computation Type	Parties Communication	Applicability	Use Cases
FHE	Any computation	Non-interactive Client–server	Available in open-source libraries	Medical data analysis Recommender systems Confidential ML
MPC	General computation (excluding specialized protocols)	Multiple clients or distributed parties	Used in practice but with limitations	Secure auctions DNA comparison Collaborative research
ORAM	Data access	Non-interactive Client(s)–server(s)	Used in secure processors and oblivious DBs	SGX integration ObliDB, Signal protocol
StE	Specific data access on encrypted structures	Non-interactive Client–server	Practical protocols for specific structures	Encrypted DBMS (e.g. MongoDB)
TEE	Any computation	Interactive Client–server with attestation service	Optional in real world cloud deployment	Data analytics Trusted AI workloads Medical Federated Learning

[20] Ivan Damgård, Kasper Damgård, Kurt Nielsen, Peter Sebastian Nordholt, and Tomas Toft. Confidential Benchmarking based on Multiparty Computation, 2015. Published: Cryptology ePrint Archive, Paper 2015/1006.

[21] David Evans, Vladimir Kolesnikov, and Mike Rosulek. *A Pragmatic Introduction to Secure Multi-Party Computation*. now, 2018.

[22] M. G. Raeini and M. Nojoumian. Privacy-Preserving Big Data Analytics: From Theory to Practice. volume 11637 LNCS, pages 45–59, 2019.

[23] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. Association for Computing Machinery. event-place: Bethesda, MD, USA.

[24] O. Goldreich, S. Micali, and A. Wigderson. How to

Table 5: Security and Performance Comparison of Privacy-Enhancing Techniques

Technique	Threat Model	Information Leakage	Performance Overhead
FHE	IND-CCA2 Adaptive attack	None by itself	High: Key Generation & Polynomial Operations
MPC	Semi-honest or malicious	Nothing beyond function output	Constant or Linear
ORAM	Semi-honest or malicious	Leakage through side-channel attacks	Logarithmic
StE	Semi-honest	Access pattern sometimes response volume	Sublinear
TEE	Malicious actor controlling server	Access patterns, plaintext in CPU	Generally near-native, bottleneck in I/O heavy

play ANY mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. Association for Computing Machinery. event-place: New York, New York, USA.

[25] Oded Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2004.

[26] S. Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Fernando Krell, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. Secure two-party computation in sublinear (amortized) time. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 513–524, New York, NY, USA, 2012. Association for Computing Machinery. event-place: Raleigh, North Carolina, USA.

[27] Marcella Hastings, Brett Hemenway, Daniel Noble, and Steve Zdancewic. SoK: General purpose compilers for secure multi-party computation. In *Proceedings - IEEE Symposium on Security and Privacy*, volume 2019-May, pages 1220 – 1237, 2019. Type: Conference paper.

[28] Yan Huang, David Evans, and Jonathan Katz. Private Set Intersection: Are Garbled Circuits Better than Custom Protocols? In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*. The Internet Society, 2012.

[29] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. On Deploying Secure Computing: Private Intersection-Sum-with-Cardinality. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 370–389, 2020.

[30] Joe Kilian. Founding crytpography on oblivious transfer. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 20–31, New York, NY, USA, 1988. Association for Computing Machinery. event-place: Chicago, Illinois, USA.

[31] Vladimir Kolesnikov. Gate Evaluation Secret Sharing and Secure One-Round Two-Party Computation. In Bimal Roy, editor, *Advances in Cryptology - ASIACRYPT 2005*, pages 136–155, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[32] Vladimir Kolesnikov and Thomas Schneider. Improved Garbled Circuit: Free XOR Gates and Applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming*, pages 486–498, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[33] Nishat Koti, Arpita Patra, Rahul Rachuri, and Ajith Suresh. Tetrad: Actively Secure 4PC for Secure Training and Inference, 2021. Published: Cryptology ePrint Archive, Paper 2021/755.

[34] Andrei Lapets, Frederick Jansen, Kinan Dak Albab, Rawane Issa, Lucy Qin, Mayank Varia, and Azer Bestavros. Accessible Privacy-Preserving Web-Based Data Analysis for Assessing and Addressing Economic Inequalities. In *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies*, COMPASS '18, New York, NY, USA, 2018. Association for Computing Machinery. event-place: Menlo Park and San Jose, CA, USA.

[35] Mengyuan Li, Yuheng Yang, Guoxing Chen, Mengjia Yan, and Yinqian Zhang. SoK: Understanding Design Choices and Pitfalls of Trusted Execution Environments. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, ASIA CCS '24, pages 1600–1616, New York, NY, USA, 2024. Association for Computing Machinery. event-place: Singapore, Singapore.

[36] Yehuda Lindell. Secure Multiparty Computation (MPC), 2020. Published: Cryptology ePrint Archive, Paper 2020/300.

[37] Yehuda Lindell and Benny Pinkas. A Proof of Security of Yao's Protocol for Two-Party Computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

[38] Yehuda Lindell and Benny Pinkas. An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. *Journal of Cryptology*, 28(2):312–350, April 2015.

[39] Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient Constant Round Multi-Party Computation Combining BMR and SPDZ, 2015. Published: Cryptology ePrint Archive, Paper 2015/523.

[40] Chang Liu, Xiao Shaun Wang, Kartik Nayak, Yan Huang, and Elaine Shi. ObliVM: A Programming Framework for Secure Computation. In *2015 IEEE Symposium on Security and Privacy*, pages 359–376, 2015.

[41] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay—a secure two-party computation system. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, page 20, USA, 2004. USENIX Association. event-place: San Diego, CA.

[42] Ueli Maurer. Information-Theoretic Cryptography. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 47–65, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[43] Moni Naor and Benny Pinkas. Computationally Secure Oblivious Transfer. *Journal of Cryptology*, 18(1):1–35, January 2005.

[44] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, pages 129–139, New York, NY, USA, 1999. Association for Computing Machinery. event-place: Denver, Colorado, USA.

[45] Claudio Orlandi. Is multiparty computation any good in practice? In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5848–5851, 2011.

[46] E. Orsini. *Efficient, Actively Secure MPC with a Dishonest Majority: A Survey*, volume 12542 LNCS of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2021. Pages: 71.

[47] Jason Perry, Debayan Gupta, Joan Feigenbaum, and Rebecca N. Wright. Systematizing Secure Computation for Research and Decision Support. In Michel Abdalla and Roberto De Prisco, editors, *Security and Cryptography for Networks*, pages 380–397, Cham, 2014. Springer International Publishing.

[48] B. Pinkas, T. Schneider, N. P. Smart, and S. Williams. Secure Two-Party Computation is Practical, 2009. Published: Cryptology ePrint Archive, Paper 2009/314.

[49] B. Pinkas, T. Schneider, C. Weinert, and U. Wieder. Efficient circuit-based psi via cuckoo hashing. volume 10822 LNCS, pages 125–157, 2018.

[50] Michael O. Rabin. How to Exchange Secrets by Oblivious Transfer. Technical Report TR-81, Harvard University: Aiken Computation Laboratory, 1981.

[51] Thomas Schneider and Michael Zohner. GMW vs. Yao? Efficient Secure Two-Party Computation with Low Depth Circuits. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography and Data Security*, pages 275–292, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[52] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979. Place: New York, NY, USA Publisher: Association for Computing Machinery.

[53] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 299–310, New York, NY, USA, 2013. Association for Computing Machinery. event-place: Berlin, Germany.

[54] Andrew C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 160–164, 1982.

[55] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167, 1986.

[56] Samee Zahur, Mike Rosulek, and David Evans. Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits Using Half Gates. In *EUROCRYPT (2)*, pages 220–250. Springer, 2015.