CYBERNETICA

JOCONDE D2.1 Technology Survey and Analysis

Technical report

Version 1.0

30.12.2024

D-16-468

Public

Project JOCONDE (Joint On-demand COmputation with No Data Exchange)
https://cros.ec.europa.eu/joconde

Project Managers:	Fabio Ricciato (Eurostat)
	Baldur Kubo (Cybernetica)

Authors (Cybernetica):	Pille Pullonen-Raudvere,
	Armin Daniel Kisand,
	Riivo Talviste,
	Kert Tali

Cybernetica AS, Mäealuse 2/1, 12618 Tallinn, Estonia.

E-mail: info@cyber.ee, Web: `https://www.cyber.ee`, Phone: +372 639 7991.

Date	Version	Description
11.10.2024	0.1	Draft for initial review
22.10.2024	0.2	Advanced draft for review
30.12.2024	1.0	Revised version for publication

Disclaimer

The field of secure private computation is very active and evolving fast, both on the side of research and practical implementations and deployments. For this reason, we plan to issue an update to this report at the end of the JOCONDE project in March 2026.

We encourage readers to provide feedback. If you notice inaccuracies, omissions, or would like to suggests corrections and improvement, please let the authors know by writing to joconde@cyber.ee. Please include sources that we can reference in future updates to this document, including e.g. academic publications, news items, software documentation, or open-source repositories.

Table of Contents

1 Introduction

The goal of this document is to introduce the main foundations of Secure Private Computation (SPC), namely Secure Multiparty Computation (MPC) and Trusted Execution Environments (TEE), that are envisioned to be relevant to the JOCONDE project. MPC uses cryptographic techniques to ensure data confidentiality in cases where multiple parties are jointly computing on sensitive data. TEE ensures security through specialised hardware. The JOCONDE project focuses on collaborative computation scenarios where (i) multiple parties make their data available as inputs to the computation (input parties), and at the same time (ii) multiple parties actually carry out the computation process (computing parties). In other words, the *collaborative*, or equivalently *multiparty* nature of the JOCONDE scenario is to be intended in the dual sense of distributed inputs and distributed computation. It is important to not assume that in JOCONDE the input parties and computing parties overlap, although they may.

This document introduces both MPC and TEE approaches and gives a thorough overview of the current state of the art. Our analysis considers the level of development and maturity of these technologies as of October 2024. We describe the strengths and advantages of these technologies and their potential limitations. This document informs the design and specification of the envisioned Multiparty Secure Private Computation-as-a-Service (MPSPCaaS) system [1]. The latter will be referenced hereafter as the JOCONDE System.

For MPC, this report covers the basics of the main methods used to build secure multiparty computation protocols. It then delves deeper into examples of concrete protocols and existing tools and services, that are the main focus of the document. We compare MPC tools based on their main purposes, security guarantees, and functionalities. Additionally, we present research where MPC and TEE are combined to improve the security guarantees of the resulting system. We also include the growing list of previous and ongoing projects that have deployed MPC solutions to work on real-world data.

For TEE, we cover the fundamentals and the main benefits of this technology for the JOCONDE project. We describe the available technologies and compare their capabilities. It is important to note that there are also many software tools that support and enable the use of secure hardware. This report also gives an overview of the relevant software components. Finally, we describe TEE adoption in the cloud.

2 Secure Multiparty Computation

Secure Multiparty Computation (MPC) refers to a set of technologies enabling the execution of collaborative computations on confidential input data without revealing any data except the final computation output. Three main methods are typically employed for ensuring secure computation: garbled circuits, homomorphic encryption, and secret sharing. All these are introduced in the following.

The participants in the computation process can act in three different roles [2]: input parties, computing parties, and output parties. Notably, the same participant may play multiple roles. The input parties make their data available as input to the computation process. The distributed computation protocol is carried out by the computing parties. At the end of the computation process, the final computation result is revealed to the output parties. A large part of the MPC literature considers the particular scenario where the computing parties act also as input parties and output parties. However, real-world deployments often require more separation between the roles, especially in the context of MPC-as-a-service solutions. Usually the set of computing parties is fixed at the beginning and does not change during the computation process, but there are also approaches like [3] that consider a dynamically changing set of parties.

While powerful theoretical results were already obtained in the 1980s, the work on practical MPC deployment started decades later. The first real-life deployment took place in 2008 with the Danish sugar beet auction [4]. Since then, the acceleration in the performance and scope of the technology has been outstanding, leading to a large number of companies working in this space. Section 2.1 below gives an overview of the core principles and parameters of secure multiparty computation. Next, Section 2.2 introduces the core secure computation methods. Section 2.3 then goes into more detail regarding the concrete versions of how the main secure computation methods are used to enable secure multiparty computation for various settings, and Section 2.4 lists existing implementations of tools that bring secure multiparty computation to life. Section 2.5 summarises the main contributions of this chapter with a comparison of the main frameworks. Finally, Section 2.6 considers cases where real data were processed by MPC applications to showcase the highest level of practical use reached insofar.

2.1 Main Properties

There are various aspects to consider when defining the security of MPC protocols. This section summarises the most prominent of these properties.

First note that discussions and formal proofs of the security of MPC protocols commonly consider a single *adversary*. The adversary, or attacker, is a generalisation of various attacks. The actions that the adversary is allowed to make without breaking the system collectively specify the security properties of the system. This descriptive approach is well in line with the real life notion of an external attacker infiltrating computation participants, either by hacking their IT system or corrupting their staff. However, the single adversary model also captures the case of an internal attacker from within the MPC setup, either acting alone or in collusion with other machines or parties.

The single adversary assumption absorbs the case of multiple adversaries acting in a coordinate manner. In fact, this represents a sort of worst case, since a set of multiple independent and uncoordinated adversaries would be collectively less powerful than a coordinated set of adversaries, or equivalently a single adversary gathering their total capabilities. In other words, demonstrating security against a single powerful adversary is sufficient to achieve security against multiple, less powerful adversaries. In the common adversary model, any party in the computation process may be corrupted by the central adversary. In practice, this covers both the cases where the party is corrupted due to its own volition or due to external infiltration.

Any MPC system displays a combination of security guarantees along the different dimensions presented below, but not all combinations of security guarantees are feasible. A comprehensive overview with several feasibility and impossibility results can be found in [5].

2.1.1 Number of Parties and Threshold

Two important parameters of any MPC system are the total number of participating computing parties and the maximum number of corrupted computing parties that can be tolerated by the system. In general, each MPC scheme sets a number of (computing) parties n and a *threshold* $t < n$ such that at least t parties must remain honest (uncorrupted) in order for any security guarantees to hold. In other words, the system is secure against up to $n-t$ corrupted (dishonest) parties.

A common division is between schemes requiring an *honest majority*, for which $t > n/2$, and schemes secure against a *dishonest majority*, for which $t < n/2$. In the latter class, the case $t = 1$, sometimes called the *full-threshold* case, is of special interest, as in such a case the system can tolerate up to $n - 1$ corrupted parties while still upholding the privacy guarantees for the remaining honest participant.

The threshold model can be generalised to more advanced access structures that specify in detail which sets of parties can be corrupted together [6]. These are most relevant for cases where the roles of the parties are not symmetric. A newer line of work also considers MPC with a dynamically changing set of parties [3]; however, this approach is not discussed further in this overview, as it is not much used for practical solutions.

2.1.2 Adversary Models

Overall, MPC considers three different kinds of adversaries: passive, active, and covert. At one extreme, the strongest *active adversary* (also known as a malicious adversary) represents an adversary who aims at breaking the system by taking any possible action against the MPC protocols. At the other extreme, the *passive adversary* (also known as semi-honest or honest-but-curious) represents an adversary that follows the protocol but tries to deduce as much as possible from the information gathered during the protocol execution. As an intermediate between these two models, the *covert adversary* model represents an adversary ready to carry out active attacks against the protocol only to the extent that such actions do not reveal him. In other words, the difference between the active and covert models is that an active adversary would try to break the protocol at all costs, while the covert adversary would abstain from taking actions that would get him caught.

A protocol is said to achieve active, passive, or covert security if it ensures privacy and integrity in the presence of the respective type of adversary. A secure computation protocol offering *active* security (also known as malicious security) preserves both privacy and integrity even in the presence of active adversaries. Conversely, a protocol providing *passive* security (or semi-honest/honest-but-curious security) ensures both the privacy and accuracy of computations, granted that all participants follow the protocol. A protocol offers covert security if the probability of detecting active actions against the protocol is reasonably high.

A separate dimension relates to whether the set of corrupted parties can dynamically change during the computation. In the *static adversary* model, the corrupted parties remain the same throughout the computation. In the *adaptive adversary* model, the adversary may corrupt more parties during the protocol execution and the decision as to whom to corrupt may be based on the computation process. Finally, the *mobile adversary* (also known as proactive security) represents the case where new parties can be corrupted and some parties may become uncorrupted during the computation. However, the latter model is not very common among current MPC frameworks.

2.1.3 Security Assumptions

Security of MPC can be based on various assumptions. The first of these is the assumption regarding the network. It is most common to assume point-to-point secure channels between participants. In other words, we exclude the network eavesdropper, which is in practice achieved using standard tools like mutually authenticated TLS channels. Some protocols also require a broadcast channel where a message can be sent so that all parties are guaranteed to receive the same message. This is achieved using a protocol for secure broadcast, e.g. [7, 8, 9], that, depending on the setting and the specific protocol, introduces its own assumptions.

Independently of the network setup, the protocol itself can achieve either *information-theoretic security* or *computational security*. In the latter case, security is linked to some computationally difficult problem, and if an efficient algorithms to solve this problem is found, the MPC protocol will be easily broken. Generally speaking, attacks against such computationally secure protocols benefit from the availability of more computational power by the attacker. In the case of information-theoretic security, the bound on security is fixed by a concrete parameter specified by some mathematical problem where the ability to solve the problem does not depend on the available computational power. For example, security can be equivalent to correctly guessing a random number from the field where the computations are executed, in which case the field size would represent the parameter that determines the security of the system.

Again, some (in)feasibility results apply to the combination of security features. It is known that information-theoretic security cannot be achieved in the general case for a dishonest majority setting [10]. This means that not all operations can be computed with information-theoretic security for a dishonest majority.

With point-to-point secure channels, information-theoretic security with guaranteed output delivery can be achieved for passive security on the assumption of an honest majority, and for active security, on the assumption that no more than a third of the parties are corrupted [10]. Conversely, with a broadcast channel, information-theoretic active security can be achieved for an honest majority [11].

Computationally secure protocols are always possible and, in particular, two-party computation can only be computationally secure.

It should be noted that the security level of the protocol is always defined by its weakest components. For example, a protocol consisting of information-theoretically secure addition and computationally secure multiplication is to be considered computationally secure overall.

2.1.4 Additional Security Properties

The previous lists covered the main properties that define the security of the MPC protocols. In addition to those, there are other aspects that should be considered but receive less explicit treatment in this document.

For models where adversary behaviour can affect the execution of the protocol, the protocols can be further classified into security with *abort*, *fair* protocols, and protocols with *guaranteed output delivery* (GOD). Active security with abort ensures that the protocol either provides an accurate output, or the participants realise that the outcome is erroneous and abort the protocol execution without producing any result. Some of such protocols allow to identify the cheating party while others do not. Fair protocols dictate that all parties, without exception, either learn the correct result or none at all. In an unfair protocol, certain parties might get the correct result while others miss out. In particular, the adversarial parties may learn the output while others do not learn anything. GOD protocols ensure that the honest parties always receive the correct result. Well-known impossibility results establish that neither fairness nor GOD properties can be achieved for certain protocol setups and functionalities. In particular, for generic functionalities, fairness and GOD are only possible for the honest-majority security model [12].

A central component of active security is *verifiability* which ensures that the honest parties can detect if the computation process or the output is incorrect. Verifiability is typically associated with the ability of participating parties to detect violations. When this ability is given to external parties, it is called *auditability* or *public verifiability* [13]. For example, the auditing approach has been proposed for the SPDZ protocol in [14]. Auditing strengthens secure computation in that it allows to discover when all parties participating to the computation are corrupted.

Protocols that have security with abort can end in a state of failure if the verification does not succeed. Active security ensures that the honest parties learn that the protocol failed. In such cases, it is preferred for the protocol to achieve *identifiable abort*, meaning that all honest parties agree that the protocol failed and they are able to identify at least one corrupted party [15]. The idea is that the honest parties could then exclude this corrupted party and attempt to perform the computation again. Identifying cheaters for protocols with dishonest majority is known to be more complicated than simply achieving protocols with abort [16]. This security notion is also extended to *completely identifiable abort* where all cheating parties are identified by the honest computing parties, and *completely identifiable auditability* where an external party can determine all cheating parties [17]. The process of identifying the misbehaving party is called *cheater detection* and can also be applied in GOD protocols.

Protocols secure against actively corrupted input parties also have to ensure *input independence*. A basic assumption in secure computation is that all input parties provide their inputs to the protocol based on true input data values. However, an actively corrupted input party may try to modify its input based on what is provided by the other parties. Note that ensuring input independence is a separate goal from ensuring privacy of the input data. For example, input independence would ensure than in a privacy-preserving auction, one party can not simply outbid another other by crafting an input to the protocol that represents $x+1$, where x is the undisclosed input from another party.

2.1.5 Relations Between Security Properties

As discussed above, components of a protocol can actually correspond to different security models and properties but the security level of the combined protocol is determined by the weakest components. Table 1 summarises the discussed properties in terms of which would be more or less preferable. It is important to note that this is a simplification and not all combinations of these properties can be achieved in one protocol. For instance, what would appear to be the strongest case, namely the combination of active security, dishonest majority, information-theoretic security, and guaranteed output delivery, is known to be generally unfeasible. On the other hand, the properties labelled as weaker tend to be relevant in practical applications and common in MPC protocols.

Often, higher security guarantees imply higher complexity, hence lower performances and scalability. However, MPC research keeps proposing ever more efficient protocols and implementations, and also conversions between the security properties (see Section 2.1.6) so that the theoretical difference in complexity is not always reflected in the practical solutions.

Table 1. Summary of MPC protocol properties

Property	Strong	Intermediate	Weak
Adversary behaviour	Active	Covert	Passive
Amount of corrupted parties	Dishonest majority		Honest majority
Output delivery	Guaranteed output delivery	Fairness	Abort
Knowledge after abort	Completely identifiable abort	Identifiable abort	No cheater detection
Corruption model	Mobile	Adaptive	Static
Security assumption	Information-theoretic		Computational

2.1.6 Converting Between Security Models

Concrete protocols proposed in the literature and implemented by frameworks specify some of the previously mentioned parameters. However, another line of work studies converting between different settings. A first natural goal is to lift passively secure protocols to deliver active security. However, keeping some of the impossibility results in mind, the conversions may need to change the threshold or security assumptions of the protocols. The classical result in [18] achieves security by incorporating *zero-knowledge proofs* in order to verify each step. Zero-knowledge proofs enable the prover to convince the verifier that a statement is true, so that the verifier only learns this fact and nothing else other than the given statement. Several other efficient approaches for converting from passive to the active security model also exist, such as [19, 20, 21].

It is possible to convert from honest majority to dishonest majority and ensure passive security of the output [22], or to convert a protocol to a corresponding publicly verifiable protocol as shown by [23]. It is also possible to extend any protocol in the preprocessing model to a protocol with cheater detection [16].

The rise of TEE technologies (see Chapter 3) has also triggered interest in the combination of MPC and TEE [24, 25], in addition to proposals to fully replace MPC protocol with TEEs [26, 27]. One possible combination approach is to create a trusted dealer for the precomputation phase (see Section 2.1.7) using TEE [28]. Another possibility is to combine MPC and TEE executions based on efficiency or level of trust [29, 30]. Another approach is to increase the security level of the MPC protocol by executing some of the computing party's code inside TEEs [31, 32]. This works particularly well when the passively secure MPC protocols lose only their integrity guarantees in the presence of active attackers, but retain confidentiality [33, 34]. Wrapping such protocols in TEEs that provide integrity can provide security against active adversaries without the need of expensive modifications to the MPC protocols themselves. In this case, the confidentiality is ensured both by the MPC protocol and TEEs, while the integrity of the computations in TEEs enables all parties to verify that others computed all their protocol messages as required. This approach to achieving active security is applied in [35]. It is also possible to achieve fairness when combining an unfair MPC protocol with TEEs [36]. Two real-world examples of combining MPC and TEEs are Signal and Coinbase [37]. It is likely that the approaches to using TEE in MPC will, in the near future, give interesting results for both the theory and practice of secure multiparty computation.

2.1.7 Online-Offline Paradigm

Many modern MPC protocols execute in what is called the *online-offline paradigm*, or equivalently the preprocessing model. The idea is that the execution of the protocol can be divided into two phases. The offline phase (or preprocessing) takes place before the input parties provide their inputs to the computation. Note that in most cases this is still an interactive protocol and requires network communication. The offline phase is used to prepare for the upcoming computation, for example by producing correlated randomness to be used later in the online phase. The online phase is the part using the actual inputs and performing the desired computation. Overall, the goal is to push as much of the expensive computations to the offline phase (precomputation) as possible, thus enabling leaner and faster execution of the online phase. A common example of execution with precomputation is a multiplication protocol using Beaver triples [38]. The precomputation itself can be considered either computation-dependent or computation-independent. In the former, the computation in the online phase can be freely chosen and the precomputation simply prepares for all possible operations. Computation-dependent precomputation can be used to efficiently prepare for the execution of some specific program or algorithm which is determined beforehand. The online-offline paradigm facilitates the decoupling of the development of the two computation phases in the MPC framework, allowing for their independent optimisation. Note that in some settings the set of parties performing the precomputation may differ from the set of parties executing the online computation phase.

Some of the following solutions are in the trusted dealer setting where the precomputation results are provided by a trusted entity and it is left up to the concrete deployment to decide how to implement this trusted third party. Very often the goal is to design the online phase with information-theoretic security and very efficient simple operations. However, the offline phase supporting this efficient online phase requires costlier computations and ensures computational security. In that case, the protocol combining the online and offline phases is computationally secure.

2.2 Core Secure Multiparty Computation Methods

This section lists the main methods used as a basis for secure computation – secret sharing, garbled circuits, and homomorphic encryption. The goal of this section is to give a high-level overview of the foundations of MPC. More details on how these are used for secure computation are presented in Section 2.3.

2.2.1 Secret Sharing

Secret Sharing (SS) is a data protection method initially developed by [39, 40] that allows the encoding of a confidential value (secret) in a set of so-called *shares*, with each share being held by a different party. Any individual share reveals nothing about the confidential value to its holder. For a perfect secret sharing scheme, at least k out of n shares are required to reveal the secret value, meaning that if k or more parties collaborate, then the secret can be reconstructed. The currently most common SS scheme is additive secret sharing, where a secret x is divided into a set of n shares $\{x_i, i = 1, \ldots, n\}$ such that $x = \sum_{i=1}^{n} x_i$. Additive secret sharing can be used for values x in any finite ring or field. This scheme implies that the reconstruction of the secret value requires the combination of all shares, i.e. $k = n$. Note that the threshold k for reconstructing the secret is related to the threshold t of honest parties to preserve MPC security guarantees presented earlier in Section 2.1.1.

In general, $t = n + 1 - k$ for computations based on a threshold secret sharing scheme. In other words, the system could be secure against up to $n - t = k - 1$ corrupted parties. However, individual computation protocols using secret sharing as a basis may lower the threshold t as the protocol itself may tolerate less corrupted parties than the secret sharing scheme.

Many SS schemes, like additive secret sharing and Shamir's secret sharing [40], are linear, implying that in addition to hiding the values they can be used for secure computation. Some of these computations, such as addition for additive secret sharing, can be performed locally by each party. These SS schemes and the local operations ensure information-theoretic security. In contrast, most computations need specialised interactive protocols, meaning that every computing party must be online throughout the protocol. These protocols may also introduce computational security assumptions. An insightful overview of SS-based secure computation is available in [41].

2.2.2 Function Secret Sharing and Homomorphic Secret Sharing

Function Secret Sharing (FSS), a specialised form of secret sharing proposed by [42], facilitates the evaluation of specific functions. The initial step involves sharing the function $f()$ amongst the parties, each receiving a share f_i of the function. If these parties select a common input x and each of them evaluates $y_i = f_i(x)$ locally, the outcome forms an additive share of $f(x) = \sum y_i$. Notably, the evaluation phase for $f_i(x)$ is non-interactive. Secure computation based on FSS often combines it with calculations using additive secret sharing, e.g. computing non-linear operations using FSS and linear layers with additive secret sharing. The functions are defined by keys, with the primary objective of ensuring the description is both succinct and conceals the function. The current literature for function secret sharing schemes is mostly focusing on the two-party case. However, multiparty versions of FSS are emerging [43, 44, 45].

Homomorphic secret sharing (HSS) [46] refers to secret sharing schemes where values are shared in a way that enables computing parties to locally evaluate some specific known function. HSS may be seen as the dual of FSS. Currently, the limiting factor of using HSS is the set of functions that can be supported [47], but their number is likely to increase and change the MPC landscape in the future.

2.2.3 Garbled Circuits

Historically the first approach to secure computation, referred to as garbled circuits (GC), is credited to [48] and further explored and formalised by [49]. This protocol allows two parties, known as the garbler and the evaluator, to securely evaluate a binary circuit. Initially, the garbler encodes the binary values (0 and 1) onto circuit wires and encrypts the circuit's truth tables. This garbled circuit, together with the input encodings, is then sent to the evaluator. The garbler can simply send the encodings corresponding to its own input bits. For the inputs of the evaluator, the garbler and sender use the oblivious transfer (OT) protocol. Oblivious transfer is a two-party protocol between a sender and a receiver where the sender has two messages and the received asks for one of them. As the output of OT, the receiver learns the desired message and the sender does not learn anything. In the OT protocol used for garbled circuits, the garbler acts as the sender and inputs the encodings of the input bits. The evaluator acts as the receiver and inputs its input bit in order to retrieve the encoding of the input bit. Using the input encodings, the evaluator can decrypt the circuit on a gate-by-gate basis. For each gate, the evaluator knows only the input encodings corresponding to the actual input values and can therefore only decrypt only the row containing the output encoding. The evaluator repeats the decryption steps until it reaches all the output gates of the binary circuit.

A significant advantage of the garbled circuits method is its universal applicability to any binary circuit, requiring only communication for circuit and key transportation. By default, the evaluation is secure against a passive garbler and an actively corrupted evaluator. The garbler is required to be passive because otherwise it could simply garble the wrong circuit. On the other hand, given an encrypted circuit, a misbehaving evaluator still could not learn extra information unless it breaks the underlying encryption scheme. The security of the protocol depends on the encryption scheme and the guarantees of the oblivious transfer protocol and therefore is computationally secure.

There are standard techniques available to optimise the circuits for garbling, such as [50, 51], along with optimisation of the garbling procedures themselves, such as the free-XOR method described in [52]. Other advancements anchor on the accomplishment of active security [53, 54].

2.2.4 Homomorphic Encryption

Encryption schemes typically define algorithms for converting a secret into a ciphertext through encryption and for retrieving the original secret through decryption. Security measures make sure that only parties holding the decryption key can decrypt the secret in the ciphertext. Homomorphic Encryption (HE) schemes extend this functionality by allowing computations to be performed on ciphertexts, with the result being a ciphertext that contains the outcome of the computation. For an asymmetric HE scheme, there is a keypair containing a public and a private key, the encryption algorithm uses the public key, whereas the decryption requires the private key.

Certain encryption schemes are homomorphic for specific operations. For instance, additively homomorphic encryption schemes like the Paillier encryption scheme [55] support addition. Fully Homomorphic Encryption (FHE) schemes, first introduced in [56], support both addition and multiplication, allowing for the evaluation of arithmetic circuits under encryption. Commonly used FHE schemes include BGV [57, 58], BFV [59, 60], FHEW [61], TFHE [62], CKKS [63], LMKCDEY [64]. Both fully and additively homomorphic encryption continue to serve as foundational structures in secure computation algorithms. For example, an additively homomorphic encryption scheme can be used to build a secure two-party multiplication protocol and used as a building block to enable multiplication of secret shared values [65]. Within current MPC schemes, multiplication protocols based on HE schemes are mostly used in the preprocessing phase to precompute multiplication operations of random values that can be used for efficient operations in the online phase. Recently, FHE has also found much use in privacy-preserving machine learning research [66, 67]. Secure computation with FHE offers passive security in the computational security model. Zero-knowledge proofs about the correctness of the computation or plaintext knowledge are used to achieve security against an active adversary.

Some HE schemes support threshold decryption. In such case the public key is known but the private key is distributed between parties during key generation. During decryption, the parties use their parts of the key to compute parts of the decryption that can be used to derive the decryption results.

Multi-key HE (MKHE) [68] is a variation of homomorphic encryption that allows parties to use different public keys for encryption while still allowing homomorphic operations with the ciphertexts. The decryption procedure requires the collaboration of all participants. Each party uses the private key they know to compute a share of the decryption and the shares are combined to produce the decryption result.

Both threshold and multi-key variants of HE enable schemes where no party alone has the power to decrypt and are therefore well suited to be used in multiparty settings. However, it is important to note that the basic model where a party defines its own keypair is also used as a building block in some MPC protocols.

2.3 Protocol Families

This section highlights some of the protocols used for secure computation. The focus here is on the theoretical ideas on how the core methods are applied to achieve protocols for different settings. They are divided to sections based on the core methods from Section 2.2 and the level of security they offer. This list of protocols is by no means exhaustive, but the aim is to cover the main approaches currently deployed by practical frameworks, especially those mentioned in Section 2.4. This document is focused on secure computation protocols that support generic computations. There are also many works targeting protocols for specific applications, like private set intersection (PSI) or oblivious RAM (ORAM), that are not considered here.

2.3.1 Additive Secret Sharing

Additive secret sharing is a version of secret sharing where each value $x \in \mathcal{R}$ in ring $\mathcal{R}$ is mapped to a set of shares $\{x_i,\, i = 1, \ldots, n\}$ such that $x = \sum_{i=1}^{n} x_i$. and the i-th party holds $x_i \in \mathcal{R}$. As all shares are needed to reconstruct the secret, privacy of the shared value is ensured as long as at least one party remains honest ($t = 1$ and $k = n$). Simple addition in $\mathcal{R}$ can be computed locally. It is also possible to devise protocols that map some other simple operation to addition (e.g. multiplication can be mapped to addition via logarithmic transformation), however in general any combination of additions and multiplications requires an interactive protocol.

The secret sharing scheme itself, as well as the local addition protocols and, therefore, computations of linear combinations of shared values are information-theoretically secure. However, depending on the setting, the multiplication protocol and other computation protocols may be computationally secure.

2.3.1.1 Passive Security

GMW. The basic GMW protocol [18] operates on the Boolean values and secret sharing. AND gates are evaluated using oblivious transfer and XOR gates are computed locally. Additive secret sharing can be seen as an extension of this idea where additions are computed locally and multiplications require a dedicated protocol that used oblivious transfer. In this case, oblivious transfer protocols work in the computational security model.

Beaver triples. The Beaver triple-based multiplication protocol [38] is especially relevant for additively shared values as there is no other generic way to compute multiplication of additively shared values. This protocol enables efficient multiplication of shared values assuming there exists a precomputed triple of secret shared values a, b and ab. This protocol idea is also the basis for the online-offline paradigm of secure computation and it is applicable to all kinds of linear secret sharing methods. The actual triple itself can be prepared in various ways, for example using the GMW protocol or homomorphic encryption.

Replicated secret sharing. In addition to Beaver triples, replication can be used to achieve multiplication using additive secret sharing. In case of replication, each party holds several of the additive shares x_i. However, this means that fewer parties are needed to fully reconstruct the secret. For example, computation using replicated secret sharing for n parties is considered in [69]. Three parties with one corruption protocol in [70] is geared to achieve high-throughput for multiplication. It extends the idea of replicated secret sharing where value v gives shares $(x_i, x_{i-1} - v)$ for $x_1 + x_2 + x_3 = 0$. The shares of 0 must be precomputed but multiplication can then be computed by each party sending just one value.

2.3.1.2 Active Security

Achieving active security with additive secret sharing requires adding some form of authentication or verification to either verify the shares or the shared value. Some examples are listed below.

GMW compiler. The first proposal showcasing that actively secure computation is possible was the GMW protocol [18] with the additional idea that its correctness can be verified using zero-knowledge proofs.

SPDZ. The SPDZ (pronounced *Speedz*) protocol was initially proposed in [71], with updates to the online computation in [72]. In addition to these variations, the precomputation method has been revised several times, a detailed overview of the developments is given in [73]. Since the latter overview, new precomputation approaches have been considered in [74, 75]. SPDZ offers active or covert security in the security with abort model for dishonest majority. However, versions with fairness, cheater detection and other properties have also been studied, see [76] for an overview.

Overall, the SPDZ protocol uses additive secret sharing in two layers: each value x is shared additively as $x = \sum x_i$ and then a message authentication code (MAC) αx is computed and also shared as $\alpha x = \sum y_i$. Each party i then holds the pair (x_i, y_i) as a share. In addition, the key α is also shared additively and each party holds one α_i. During the publishing operation, the parties verify that the MAC of the published output is computed correctly. The protocol works for any number of $n \geq 2$ parties and offers active or covert security so that up to $n - 1$ parties can be corrupted. All values and shares are in some finite field, the size of the field defines the security level, as overall the protocol is as secure as hard it is to guess the random MAC key α.

The protocol works in the precomputation paradigm. At the minimum, the precomputation phase produces Beaver triples for multiplication, shares of random values for inputs, and the shared MAC key. The core of the online phase supports linear combinations and multiplication. However, various algorithms have been developed for this version. There is also a version of SPDZ with function-dependent preprocessing proposed in [77].

SPD$\mathbb{Z}_{2^k}$. SPD$\mathbb{Z}_{2^k}$ [78, 79] (pronounced *Speedz 2k*) is a variation of the SPDZ protocol for the case of rings, specially computation modulo 2^k. Overall, the protocol and share representation are very similar to that of SPDZ; however, extra care is needed in analysis and fixing needed share sizes to account for zero divisors in the ring and maintain the desired level of security.

TinyTables. TinyTables [80] is a two-party secret sharing-based protocol for Boolean values in the preprocessing model. Active security is achieved by adding a MAC to the shared value. The preprocessing model provides a multiplication triple used to evaluate AND gates, while XOR and NOT gates are non-interactive. A passively secure version of the protocol is implemented in FRESCO (see Section 2.4.16). The idea is to generate lookup tables for efficient computation in the online phase.

TinyOT. TinyOT [81, 82, 83] uses MACs similar to SPDZ but is focused on using oblivious transfer for computations. It has affected the precomputation of SPDZ, SPD$\mathbb{Z}_{2^k}$ and TinyTables and been extended to various other protocols, see [83]. This protocol leverages OT extensions to produce authenticated bits which serve as a basis for the rest of the computations.

2.3.2 Shamir's secret sharing

In Shamir's secret sharing scheme [40] a secret v is shared as evaluations of a polynomial $f(x) = a_{t-1}x^{t-1} + \ldots a_1 x + v$. Each party learns $f(x_i)$ for a publicly known x_i. The party computing the shares picks the $t - 1$ random coefficients a_i, meaning that t evaluation points are necessary to learn the secret v. Addition operations can be computed by each party locally. Multiplication can be evaluated for honest majority, i.e. $t < n/2$.

2.3.2.1 Passive Security

BGW protocol. The BGW protocol [10, 84, 85] uses Shamir's sharing with honest majority to provide information-theoretically secure MPC. It follows the blueprint that all parties share their inputs and all operations are computed using secret shares. Additions can be computed locally and multiplications are computed collaboratively. Each party multiplies their respective shares and then secret shares the result between all parties. The parties then do some local computation to derive the correct shares for the multiplication output.

2.3.2.2 Active Security

For thresholds less than $n/3$, Shamir's secret sharing can be used with active security so that honest parties can verify correctness thanks to replication using the BGW protocol [10]. More generally, this applies to all multiplicative linear secret sharing schemes [86].

2.3.3 Function Secret Sharing

2.3.3.1 Passive Security

Function secret sharing with preprocessing is considered in [87, 88]. The dealer picks a random mask for each of the wires in a circuit. This mask is used to hide the real values in the evaluation. Both computing parties see the masked value. FSS is used the evaluate the computation gates, both parties input the masked input wire to the functionality and learn the masked output as a result. The dealer is the one generating the FSS for each of the gates. It can be seen as a generalisation of the TinyTables protocol. The dealer can be replaced by a circuit-dependent preprocessing phase. This can be extended to circuit-independent case. Similarly, this can be extended to multiparty computation and active security.

2.3.3.2 Active Security

Pika [89] uses FSS for an honest-majority actively secure protocol with abort in order to compute non-linear functions with constant rounds. Achieving active security requires ensuring that FSS keys are well formed. This protocol builds on the lookup table ideas from TinyTables protocol.

2.3.4 Garbled Circuits

Note that the general idea of garbling is secure against an actively corrupted evaluator but only secure against a passively secure garbler. The following distinguishes methods based on the corruption of the garbler.

2.3.4.1 Passive Security

For passively secure garbled circuits the main question is to enhance the efficiency of the garbling either by choosing suitable encryption methods or by reducing the amount of data needed to encrypt the gates.

Free XOR. The idea of the Free-XOR method [52] is that instead of encrypting XOR gates, the new label is computed directly from the labels of the inputs. Hence, XOR gates do not require any cryptographic operations to garble or to evaluate.

Garbled row reduction. The garbled row reduction idea [90] proposes methods to not send either one or two rows of the full garbled circuit. The reduction of one row can be combined with the free XOR idea.

Half gates. The half gates approach [91] enables garbling XOR gates for free and sending only two times the length of the ciphertext for each of the AND gates. The cited paper also establishes that this result is optimal for linear garbling.

Three-halves. The three-halves approach [92] proposes a way to slice each wire label in half; different computations can be used to derive the halves. This slicing allows them to bypass the lower bound by [91] and achieves a lower bound for a more general class of garbling operations [93]. In addition, this approach is compatible with the free XOR method.

Stacked garbling. Stacked garbling [94, 95] improves garbling of functions with conditional branching. The number of ciphertexts needed in the garbling is proportional to the longest execution path rather than the whole circuit.

Multiparty garbling. The two-party garbled circuits approach is lifted to a multiparty protocol in the BMR protocol [96]. The core idea is that an MPC protocol is used to generate the garbling and then all parties evaluate the circuit on their own. The free XOR version of the BMR protocol is described in [97].

2.3.4.2 Active Security

Ensuring security against an actively corrupted garbler requires verifying that the circuit is actually correctly garbled.

Cut-and-choose. The idea of the cut-and-choose method is that the garbler generates many circuits and sends these to the evaluator. The evaluator chooses a subset of these to be opened and verifies that these are correct. If all verified circuits are correct, the evaluator evaluates all the unopened circuits and uses the majority output as the output of the computation. This approach was proposed by [98] with various works improving on specific parameters.

LEGO. The idea of the Large Efficient Garbled-circuit Optimisation (LEGO) [53, 99, 100] protocol is to do the cut-and-choose at the gate level and then solder these together to fault-tolerant buckets for each gate. Versions of this approach work with free XOR and row reduction optimisations.

DUPLO. DUPLO [101] merges the whole circuit cut-and-choose and the LEGO approaches, allowing to perform cut-and-choose for arbitrary subcircuits.

Multiparty garbling. Garbled circuits is usually a two-party protocol; however, there exist several approaches to lift this to the multiparty setting following the BMR approach. Recent work [102] achieved a communication efficient multiparty garbling scheme for honest majority where communication complexity decreases as the number of parties increases.

Authenticated garbling. The authenticated garbling approach [103, 104, 105] generates authenticated secret shared wire labels and executes the semi-honest version of garbled circuits protocol, after which the evaluator can verify the correctness of each evaluated AND gate. In essence, it combines actively secure authenticated secret sharing with passively secure garbled circuits. The authenticated garbling protocol [106] combines the ideas of [103] and TinyOT. Authenticated garbling can also be used for multiparty garbling.

2.3.5 Homomorphic Encryption

The basic idea of using fully homomorphic encryption for MPC is that all parties encrypt their inputs with a threshold encryption scheme, the result can be computed publicly and then all parties carry out the threshold decryption. This offers passive security. Active security requires ensuring that that inputs are formed correctly and the decryption is computed correctly. However, concrete proof methods need to be tailored for the protocols in order to achieve efficiency.

2.3.5.1 Passive Security and Semi-Malicious Security

In the semi-malicious model, the adversary can pick its own randomness but all messages have to be computed following the protocol using this randomness [107]. This model thus lies between the passive and active security models.

Multi-Key HE. The benefit of using Multi-key HE (MKHE) for secure computation is that each joining party can set up its own keypair. It can especially be used to create round-efficient MPC protocols. For example, [108] obtains a two-round MPC protocol where active security is achieved with non-interactive zero-knowledge proofs. The setup requirements for this approach are further studied in [109] with extra rounds added to compute the setup. A three-round protocol without setup is developed in [110]. Note that the latter is secure in the semi-malicious model. MKHE is sometimes also called multiparty HE.

Threshold FHE. A threshold version of BFV is used in [111] to achieve a passively secure dishonest majority MPC protocol. This work generalises the approaches of threshold and multi-key FHE schemes and proposes an efficient MPC computation method based on the efficiency advances of FHE schemes.

Multi-Group HE. A generalisation of MKHE and threshold HE called multi-group HE is derived in [112]. In this case, a joint key is generated like in the threshold HE case but homomorphic operations are allowed also with ciphertexts obtained using different keys like in MKHE. This is then used to propose MPC for semi-malicious adversaries with dishonest majority.

2.3.5.2 Active Security

Threshold FHE. A robust MPC protocol with active security is proposed in [113]. This protocol allows for at most third of the computing parties to be corrupted. The goal of this work is to use efficient FHE components like threshold FHE to obtain a programmable MPC protocol that allows intermediate publishing. A focus is also on evaluating a pseudorandom function under encryption. This protocol uses transciphering to reduce the overhead that large FHE ciphertexts and zero-knowledge proofs add to the storage of secure data. In transciphering, the (usually symmetric) encryption is computed using the homomorphic properties. It can be used to store long term data or parties can encrypt their inputs using the symmetric encryption scheme and only encrypt the key using the FHE scheme. The computing parties can then transform the inputs to FHE as needed.

PELTA. PELTA [114] focuses on proposing mechanisms to protect threshold, multi-key, and multi-group HE schemes against an active adversary. They combine commitments and zero-knowledge proofs to verify correctness of each computation local step and propose another proof for proving correct aggregation. This is geared towards lattice-based commitments, encryption and proof systems.

Verifiable computation. In the verifiable computation model, the computing machine produces a proof of correct computation. One way to achieve this is using zero-knowledge proofs. Recent work [115] extends this by combining homomorphic succinct non-interactive argument of knowledge (SNARK) with hashing to prove correctness of the homomorphic evaluation. The limitation of such approaches is that it only supports limited depth computations because the verification does not apply to relinearisation.

2.3.6 Mixed-mode protocols

The BMR protocol described for multiparty garbling is already an example of a protocol combining several ideas. However, various other combinations can be considered. This section lists some more established and promising approaches. In general, the term mixed-mode protocol (or hybrid protocol) refers to any computation allowing a protocol to be represented as different modes of computation, either a combination of arithmetic and Boolean or a combination of secure computation techniques. It is a whole line of research of its own to derive efficient mixed-mode protocols, a good overview is given in [116]. It is also a separate line of work to analyse the cost of the protocols and compile a specific operation to the most efficient MPC protocol [117].

2.3.6.1 Arithmetic-Boolean-Yao

Arithmetic-Boolean-Yao (ABY) [118, 119] is a two-party passively secure framework for combining additive secret sharing over rings and Booleans with the garbled circuits computation. Its strength is proposing efficient protocols to convert from one data representation to another. Its extension ABY^3 [120] works for three parties with one actively corrupted party. ABY itself inherited many ideas from TASTY [121].

2.3.6.2 Secure Multiparty Computation and Trusted Execution Environments

Homomorphic encryption and trusted execution environments are combined in [122] to enable a system that achieves security against actively corrupted clients and servers for a machine learning application. The privacy of the clients is ensured by HE and the service providers protect their models using TEE even when using public services to host the computation. TEEs are also used to provide integrity of the computation. However, using HE for client inputs also means that the clients do not need to trust TEEs with their privacy. Similar combination to use MPC for privacy and TEEs for integrity is taken by [35].

2.3.6.3 Function and Additive Secret Sharing

It is possible to combine FSS and SS so that FSS is applied over secret shared inputs to produce shared outputs as in [87]. The parties would reconstruct a masked input based on the shared input and use this as an input to the FSS scheme. This way, FSS can be used to compute the parts of the computation that are otherwise complicated to be computed with secret shares.

2.3.6.4 Garbling Mixed Circuits

An extension of the garbled circuits approach focuses on garbling circuits that combine Boolean operations and arithmetic operations as well as bit-decomposition of the arithmetic values [123]. This line of work was started by the idea of garbling arithmetic circuits [124].

2.4 Secure Computation Tools

The previous section focused on the core technology and protocols. This section lists some common tools and frameworks for secure multiparty computation in alphabetical order. The goal is to cover main historical approaches as well as currently active developments. Note that the amount of information available for each of the tools varies; the goal of this section is thus to simply list relevant tools and provide references. A summary and comparison of these tools can be found in Section 2.5. In fact, this section can be seen as serving as a glossary for Section 2.5. Note that this list is not exhaustive: most importantly, MPC solutions clearly geared towards blockchain applications or specific machine learning tasks are not included as the focus of this report is on data analysis and more general frameworks.

2.4.1 Asterisk

Asterisk [125][1] explores achieving active security with dishonest majority with a semi-honest helper party. The helper party also allows this setting to achieve fairness. Secure computation is based on authenticated additive shares. This implementation is focused on large-scale secure computation, with benchmarks showing 100-party computation.

2.4.2 Carbyne Stack

The Carbyne Stack project[2] is an effort to create a technology stack supporting the deployment of MPC applications in a cloud-native manner. Carbyne Stack is a platform-based approach to MPC seeking to create a well-integrated environment supplementing the interfacing needs of the actual MPC runtime (i.e. the program executing MPC protocols). Its key offerings include orchestration, like provisioning and discovery of MPC processing nodes within Kubernetes clusters, storage for secret data, and MPC offline phase facilities. Altogether, it forms an ecosystem with its own client applications and infrastructure automation to easily deploy and use MPC.

In essence, Carbyne Stack could wrap any MPC runtime which can operate in the secure outsourcing model – i.e. where data is provided to a fixed set of computing parties to run the computation on behalf of the clients – and can accommodate the necessary interfaces. At the time being, only MP-SPDZ (Section 2.4.28) is supported, with an integration roadmap ready for incorporating the Sharemind MPC protocol core (Section 2.4.43) [126]. Work has also been done towards integrating Secrecy[3] (Section 2.4.37).

This open source project is an initiative launched by Bosch Research which has led the development publicly since September 2021. However, once Carbyne Stack has gotten more traction, they are considering transitioning to a community-based maintenance and decision-making

[1]Asterisk https://github.com/cris-coders-iisc/Asterisk Last accessed: October 2024
[2]Carbyne Stack. https://carbynestack.io/ Last accessed: October 2024
[3]Bosch Tube: John Liagouris. *Secrecy: Secure collaborative analytics in untrusted clouds*, 2022. https://bosch-ext.mediaspace.de.kaltura.com/media/SecrecyA+Secure+collaborative+analytics+in+untrusted+clouds/0_j66zkevf Last accessed: October 2024

model to provide a common neutral platform for different MPC deployments[4]. Independent contributors currently include Honda Research and University of Technology Sydney among others.

2.4.3 CBMC-GC

CBMC-GC [50, 127][5] is a compiler to get Boolean circuits from a subset of ANSI-C. It outputs circuits in formats suitable for TinyGarble (Section 2.4.50) and Fairplay (Section 2.4.13). It also included tools to run the protocols using ABY.

2.4.4 CipherCompute

CipherCompute[6] is based on SCALE-MAMBA (Section 2.4.35) and is a free version of the Cosmian Collaborative Confidential Computing tool[7]. Applications for CipherCompute can be developed in Rust.

2.4.5 CirC

CirC[8] stands for Circuit Compiler and is a tool for compiling high-level languages to circuits for various tools. Silph [128][9] is a variation of CirC for generating mixed-mode MPC protocols using two-party ABY as a backend but is intended to support more backends.

2.4.6 COMBINE

COMBINE [129][10] (COMpilation and Backend-INdependent vEctorisation) proposes backend-independent compilation and optimisation of secure computation protocols. They focus on optimising SIMD operations. COMBINE currently supports MOTION (Section 2.4.27) and MP-SPDZ (Section 2.4.28) as backends.

2.4.7 Conclave

Conclave [130][11] is a query compiler for MPC. The goal of Conclave is to separate the computation to parts that can be executed publicly and parts needing an MPC backend. Conclave development was led by Boston University but is no longer active. It uses Oblic-C (Section 2.4.30) and Sharemind MPC (Section 2.4.43) frameworks as secure computation backends.

Note that there is also Conclave platform[12] for trusted execution environments for secure data collaboration across multiple parties. It was developed by R3 but has been discontinued.

[4] Bosch Tube: Dr. Sven Trieflinger. *Towards an Ecosystem for Open Cloud-Native Secure Multiparty Computation*, 2023. https://bosch-ext.mediaspace.de.kaltura.com/media/Towards+an+Ecosystem+for+Open+Cloud-Native+Secure+Multiparty+Computation/0_3yk5mxli Last accessed: October 2024
[5] GBMC-GC https://gitlab.com/securityengineering/CBMC-GC-2 Last accessed: October 2024
[6] CipherCompute https://github.com/Cosmian/CipherCompute Last accessed: October 2024
[7] Cosmian https://cosmian.com/ Last accessed: October 2024
[8] CirC https://github.com/circify/circLastaccessed:October2024
[9] Silph https://github.com/edwjchen/Silph Last accessed: October 2024
[10] COMBINE https://github.com/milana2/ParallelizationForMPC Last accessed: October 2024
[11] Conclave https://github.com/multiparty/conclave Last accessed: October 2024
[12] R3 Conclave https://github.com/R3Conclave Last accessed: October 2024

2.4.8 Demeter

Demeter is a MPC-as-a-service platform by Partisia Blockchain[13]. It uses a MPC protocol called REAL[14]. REAL is based on Boolean values and support for arithmetic protocols is in its roadmap. Details for the implementation of this protocol are not publicly available. Partisia is part of a project to create a privacy-preserving platform for health data in Denmark[15] but information about their protocols or frameworks is not available.

2.4.9 Divvi Up and Prio

Divvi Up[16] it a system for privacy-preserving aggregate statistics used for collecting telemetry data. It is based on Prio [131][17], an actively private framework for computing aggregate statistics. Privacy of the inputs is ensured as long as one participant is honest; on the other hand, correctness is only ensured if all computing parties are honest. The idea is that the client sends the query and generates the precomputed values for the computing parties to use. Divvi Up is developed by the Internet Security Research Group (ISRG) and used by Mozilla Firefox [18] and Horizontal[19]. A fork of Divvi Up is used by Tinfoil for browser data collection[20].

2.4.10 EMP

Efficient Multi-Party Computation (EMP) toolkit[21] implements garbled circuits with different security models. For example, the toolkit also includes a multiparty authenticated garbled circuit protocol [106] for malicious security. EMP has been used for clinical research to power VaultDB [132].

2.4.11 EasySMPC

EasySMPC [133][22] is a no-code tool simplifying the practical application of secure multiparty computation. Its secure computation is based on additive secret sharing and the arithmetic extension of GMW protocol for the passive security case. The goal of EasySMPC is to demonstrate that MPC can be made easy to use and deploy and it currently only supports computing sums.

2.4.12 EzPC

EzPC [134][23] (pronounced *easy-peasy*) offers a C-like language for secure machine learning with a compiler and several secure computation frameworks. EzPC supports secret sharing, garbled circuits, and function secret sharing. Its initial version used ABY as a backend but it now contains

[13] Demeter https://partisiablockchain.com/demeter-mpc-as-a-service/ Last accessed: October 2024
[14] REAL protocol https://partisiablockchain.com/roadmap-spotlight-3-arithmetic-mpc-real-protocol/ Last accessed: October 2024
[15] Oscar project https://www.oscar-project.com/about Last accessed: October 2024
[16] Divvi Up https://divviup.org/, source code in https://github.com/divviup Last accessed: October 2024
[17] Prio https://crypto.stanford.edu/prio/ Last accessed: October 2024
[18] Divvi Up blog. *Divvi Up is providing privacy-preserving metrics for Firefox* 2023 https://divviup.org/blog/divvi-up-in-firefox/ Last accessed: October 2024
[19] Divvi Up blog. *Bringing Privacy-respecting Telemetry to Human Rights Defenders* 2023 https://divviup.org/blog/horizontal/ Last accessed: October 2024
[20] Divvi Up and Tinfoil software https://github.com/divviup/janus Last accessed: October 2024
[21] EMP https://github.com/emp-toolkit Last accessed: October 2024
[22] EasySPMC https://github.com/easy-smpc/easy-smpc Last accessed: October 2024
[23] EzPC repository https://github.com/mpc-msri/EzPC, project page https://www.microsoft.com/en-us/research/project/ezpc-easy-secure-multi-party-computation/ Last accessed: October 2024

several of its own secure computation implementations. Notably, its Aramis component uses trusted hardware to lift passively secure protocols to active security and GPU-MPC runs function secret sharing accelerated on GPUs. It supports protocols for two and three parties. EzPC supports signed and unsigned integers as well as Boolean values. EzPC is developed by Microsoft Research India.

2.4.13 Fairplay

Fairplay [135] was one of the first MPC frameworks, demonstrating a two-party implementation of garbled circuits. FairplayMP [136][24] extend this to an n-party setting by implementing a BMR protocol with the setup from honest-majority BGW protocol. Both versions of Fairplay supported a language called SFDL to program the secure computation. It is no longer being maintained.

2.4.14 FANNG-MPC

FANNG-MPC [137][25] is a framework for actively secure MPC. It is geared towards machine-learning-as-a-service applications with secure two-party computation. It is a successor of SCALE-MAMBA (Section 2.4.35) that is data focused, adds support for databases, and contains many optimisations toward machine learning. It considers the online-offline paradigm for mixed-mode protocols. Its goal is to decouple these phases and allow conversion from different precomputed values to online protocol, for example MPC precomputation for two-party online phase. Precomputation of triples is using FHE, bits is using OT and also circuit garbling can be computed in preprocessing. FANNG supports Boolean, integer and fixed point arithmetic. FANNG is developed by Cryptography Research Centre in Technology Innovation Institute in Abu Dhabi.

2.4.15 FBPCP

The Facebook Private Computation Platform (FBPCP)[26] uses the FBPCF [138][27] framework for secure two-party computation. The framework uses a mixed-mode EMP toolkit (Section 2.4.10) and Boolean secret sharing based on GMW while also implementing ORAM. No new features are currently being added to this project. It is developed by Meta Research and has been used to analyse ad performance[28].

Note that Meta is also behind the development of a research tool for privacy-preserving machine learning called CrypTen [139][29].

2.4.16 FRESCO

FRamework for Efficient and Secure COmputation (FRESCO)[30] is a computation framework that supports the TinyTables protocol for passive security and SPDZ and SPD$\mathbb{Z}_{2}^{k}$ protocols for active security. FRESCO supports fixed-point, Boolean, and integer computations and implements sup-

[24]FairplayMP https://github.com/FaiplayMP/FairplayMP Last accessed: October 2024
[25]FANNG-MPC https://github.com/Crypto-TII/FANNG-MPC Last accessed: October 2024
[26]FBPCP https://github.com/facebookresearch/fbpcp Last accessed: October 2024
[27]FBPCF https://github.com/facebookresearch/fbpcf Last accessed: October 2024
[28]Meta news. *What Are Privacy-Enhancing Technologies (PETs) and How Will They Apply to Ads?* 2021 https://about.fb.com/news/2021/08/privacy-enhancing-technologies-and-ads/ Last accessed: October 2024
[29]CrypTen https://crypten.ai/ Last accessed: October 2024
[30]FRESCO documentation https://fresco.readthedocs.io/, source https://github.com/aicis/fresco Last accessed: October 2024

port for external input and output parties. FRESCO supports Boolean circuits in the Bristol format[31]. FRESCO is developed by Alexandra Institute. It has been used for financial benchmarking [140].

2.4.17 Frigate

Frigate [141] is a compiler from a C-like language to circuits. It supports the DUPLO protocol for garbled circuit evaluation.

2.4.18 FudanMPL

FudanMPL[32] offers various tools for secure multiparty computation for machine learning. They have implementations of homomorphic encryption and garbled circuits, and they are also using MP-SPDZ (Section 2.4.28). FudanMPL is developed by the Data Security and Governance Research Group at Fudan University.

2.4.19 Helium

Helium [142][33] is a secure computation platform based on multiparty homomorphic encryption. Helium is built using Lattigo (Section 2.4.24). It achieves passive security.

2.4.20 FUSE

FUSE [143][34] is a framework for unifying and optimising secure computation implementations. It is a layer between the compiled circuit and secure computation execution. FUSE supports HyCC (Section 2.4.22), Bristol circuit format and MOTION (Section 2.4.27) circuits as inputs that can be converted to FUSE representation. FUSE then uses MOTION and MP-SPDZ as secure computation backends for executing the computation. The goal of FUSE is to ensure efficient storage for the circuits and enable interoperability of MPC compilers and computation frameworks. FUSE is developed by the Cryptography and Privacy Engineering Group at TU Darmstadt.

2.4.21 HybrTC

HybrTC [30] considers a hybrid approach between MPC and TEE execution allowing the participants to pick an execution scheme depending on which level of trust they are willing to put to the guarantees given by TEEs. On the MPC side, HybrTC supports BMR protocol for garbled circuits, secret sharing with BGW protocol and homomorphic encryption. Any of the computing parties can be actively corrupted on the assumption that the adversary can not access the root key used for the TEEs. The input and output parties can be passively corrupted. The input and output parties can be semi-honestly corrupted.

[31] Bristol format `https://nigelsmart.github.io/MPC-Circuits/` Last accessed: October 2024
[32] FuranMPL `https://github.com/FudanMPL` Last accessed: October 2024
[33] Helium `https://github.com/ChristianMct/helium` Last accessed: October 2024
[34] FUSE `https://github.com/encryptogroup/FUSE` Last accessed: October 2024

2.4.22 HyCC

HyCC compiler [144][35] compiles C to a mixed-mode MPC protocol for GMW and garbled circuits protocols. Outputs of HyCC can be evaluated by ABY and MOTION (Section 2.4.27). HyCC is based on GBMC-GC (Section 2.4.3).

2.4.23 JIFF

JIFF[36] is a Javascript library tailored for MPC protocol prototyping and development of privacy-preserving web applications. It provides passive security with secret sharing. JIFF is developed by Boston University. It has been used for a Boston wage cap study [145].

2.4.24 Lattigo

Lattigo[37] is a lattice-based multiparty homomorphic encryption library written in Go. It implements BFV, BGV, and CKKS in their multiparty versions. It is used to implement [112] and the PELTA protocol. Lattigo is developed by Tune Insight.

2.4.25 LIBSCAPI

Libscapi[38] (where SCAPI means Secure Computation API) is a secure computation API in C++ for implementing both two-party and multiparty computation. It has been used for various research systems for garbled circuits, replicated sharing based computation, the GMW protocol, and more[39]. Libscapi is developed by Bar Ilan University.

2.4.26 Manticore

Manticore [146, 147] combines passively secure additive secret sharing and garbled circuits with half-gates and free XOR optimisations. The protocol has passive security with dishonest majority. Manticore operates in the online-offline paradigm where the offline phase is carried out by a trusted dealer or by the computing parties using an interactive protocol. In the trusted dealer model, the trusted party is the garbler and the computing parties all evaluate the garbled circuits. Manticore supports fixed-point and Boolean arithmetic.

2.4.27 MOTION and ABY

MOTION [148][40] is a framework for mixed-protocol secure computation. It combines arithmetic and Boolean GMW over rings and OT-based BMR protocol [97], introducing conversions between the protocols. MOTION uses integer and Boolean values. This results in a dishonest-majority passively secure framework. MOTION implements asynchronous evaluation where each gate in the circuit can be computed as soon as its inputs are available. MOTION uses HyCC (Section 2.4.22) to compile C code to MPC protocols.

[35] HyCC `https://gitlab.com/securityengineering/HyCC` Last accessed: October 2024
[36] JIFF `https://github.com/multiparty/jiff` Last accessed: October 2024
[37] Lattigo `https://github.com/tuneinsight/lattigo` Last accessed: October 2024
[38] Libscapi `https://github.com/cryptobiu/libscapi` Last accessed: October 2024
[39] MPC benchmarking tool `https://github.com/cryptobiu/MPC-Benchmark` Last accessed: October 2024
[40] MOTION `https://github.com/encryptogroup/MOTION` Last accessed: October 2024

MOTION2NX [41] extends MOTION by adding different two-party passively secure protocols. MOTION-FD [149][42] enhances MOTION with function-dependent preprocessing for honest-majority three-party protocols. MOTION-FD also supports fixed-point computation.

MOTION is also a successor of ABY (Section 2.3.6.1)[43], especially implementing ideas from [119]. However, the implementation of ABY^3[120][44] is independent. Both MOTION and ABY are developed by Cryptography and Privacy Engineering Group at TU Darmstadt.

2.4.28 MP-SPDZ

Multi-Protocol SPDZ (MP-SPDZ) [150][45] is a framework to compare and benchmark MPC protocols. It has implementations of a vast set of protocols for different security models and computation methods, including SPDZ, SPD$\mathbb{Z}_{2^k}$, BMR, and replicated secret sharing. All these protocols can be used with a common programming interface. Programs in MP-SPDZ are written in Python and compiled into bytecode specific to the MP-SPDZ virtual machine. The standard library supports operations on secret integers, fixed-point numbers, and floating-point numbers.

A notable feature of MP-SPDZ is support for a subset of the Keras interface for private neural network training and evaluating. Also, an ORAM protocol enables accessing arrays with a secret index. MP-SPDZ does not support database operations, or table joins, and all application data must fit into memory.

MP-SPDZ is developed by CSIRO Data61 Engineering & Design.

2.4.29 MPyC

Multiparty Computation in Python (MPyC)[46] is a framework for passively secure n-party computation with honest majority with Shamir's secret sharing in the BGW protocol. MPyC is a follow-up to VIFF (Section 2.4.52). Protocols for MPyC can be written in Python and the goal is to mimic Python built-in functions with an MPC backend. MPyC supports integer, fixed-point, floating-point, and finite field arithmetic. It also supports group arithmetic used for threshold cryptography. MPyC also supports lists and arrays, including lists with oblivious access.

A variation of this framework called hMPC[47] is also implemented in Haskell. An extension to verifiable version of MPyC that uses zero-knowledge proofs is in development[48]. MPyC has also been ported for use in web browsers[49].

MPyC has been used by TNO to develop a privacy-preserving system to predict heart failure[50].

[41]MPOTION2NX https://github.com/encryptogroup/MOTION2NX Last accessed: October 2024
[42]MOTION-FD https://github.com/encryptogroup/MOTION-FD Last accessed: October 2024
[43]ABY https://github.com/encryptogroup/ABY Last accessed: October 2024
[44]ABY^3 https://github.com/ladnir/aby3 Last accessed: October 2024
[45]MP-SPDZ https://github.com/data61/MP-SPDZ Last accessed: October 2024
[46]MPyC documentation https://www.win.tue.nl/~berry/mpyc/, source code https://github.com/lschoe/mpyc Last accessed: October 2024
[47]hMPC https://github.com/nickvgils/hMPC Last accessed: October 2024
[48]Verifiable MPyC https://github.com/toonsegers/verifiable_mpc/ Last accessed: October 2024
[49]MPyC for web applications https://github.com/e-nikolov/mpyc-web Last accessed: October 2024
[50]Marie Beth van Egmond. *Identifying heart failure patients at high risk using MPC* 2020 https://medium.com/applied-mpc/identifying-heart-failure-patients-at-high-risk-using-mpc-ab8900e75295 Last accessed: October 2024

2.4.30 Obliv-C

Obliv-C secure computation compiler[51] is a C extension for two-party garbled circuit execution. It adds an `obliv` qualifier to C types. It was used to build the Absentminded Crypto Kit[52] to support MPC. Obliv-C was used to design a system to match students to sororities [151]. The library is no longer being maintained.

2.4.31 ObliVM

ObliVM [152][53] provides a Java-like language for programming secure computation for a backend using garbled circuits. ObliVM is no longer being maintained.

2.4.32 OpenFHE

OpenFHE [153][54] is an open source project and initiative for implementing post-quantum fully homomorphic encryption. OpenFHE contains all the capabilities of the PALISADE[55] library for homomorphic encryption. In addition, it contains features of HElib [154][56] and HEAAN[57]. It implements BGV, BFV, CKKS, LMKCDEY and FHEW schemes. It also contains threshold variants of BGV, BFV and CKKS for multiparty use. In addition, it supports switching between some of the encryption schemes and computing more complex operations. OpenFHE is extended by Intel HEXL Acceleration[58] to accelerate some of its operations.

OpenFHE is used by Duality to build their Duality Query Engine for secure data collaboration[59].

2.4.33 PICCO

PICCO (Private dIstributed Computation COmpiler) [155, 156][60] is a compiler for an extension of C to a Shamir's secret sharing-based MPC protocol. It supports passive security for an arbitrary number of parties with honest majority. Another implementation that corresponds with the PICCO formalisation is also available[61]. PICCO supports Boolean, integer, and floating-point computations.

[51] Obliv-C https://github.com/samee/obliv-c/ Last accessed: October 2024
[52] Absentminded Crypto Kit https://bitbucket.org/jackdoerner/absentminded-crypto-kit/ Last accessed: October 2024
[53] ObliVM https://github.com/oblivm Last accessed: October 2024
[54] OpenFHE initiative https://www.openfhe.org/, source code https://github.com/openfheorg Last accessed: October 2024
[55] PALISADE FHE library https://palisade-crypto.org/ Last accessed: October 2024
[56] Helib https://github.com/homenc/HElib Last accessed: October 2024
[57] HEAAN https://heaan.it/ source code https://github.com/snucrypto/HEAAN Last accessed: October 2024
[58] Intel HEXL Acceleration for OpenFHE https://github.com/openfheorg/openfhe-hexl Last accessed: October 2024
[59] Duality Query Engine by Duality https://dualitytech.com/platform/duality-query/ Last accessed: October 2024
[60] PICCO https://github.com/applied-crypto-lab/picco Last accessed: October 2024
[61] Alternative implementation of PICCO https://github.com/applied-crypto-lab/formal-picco Last accessed: October 2024

2.4.34 PySyft, SyMPC and Sycret

PySyft[62] is a framework for privacy-preserving machine learning and data science.

SyMPC[63] is a library extending PySyft with secure multiparty computation features. It supports the ABY^3 and Falcon [157] protocols for both active and passive security, and FSS and SPDZ computation for passive security. The focus of the project is on training and evaluating neural networks.

Sycret[64] is a function secret sharing library with Rust backend used in PySyft. It also makes use of AES-NI hardware acceleration. It is not yet production ready.

PySyft and related tools are part of the OpenMined open-source software for privacy-preserving data analysis. They have been used by the United Nations PET Lab to analyse international trade[65].

2.4.35 SCALE-MAMBA

The framework called Secure Computation Algorithms from LEuven Multiparty AlgorithMs Basic Argot (SCALE-MAMBA)[66] started out as a follow-up to earlier SPDZ protocol developments but later was expanded to support computations with various secure computation methods. For example, it combines linear secret sharing and garbled circuits. Applicatio programming is done in Rust. Some implemented protocols provide support for data types such as 64-bit integers, integers modulo a prime, fixed-point numbers, and IEEE floating point numbers. The standard library contains common math operations, oblivious memory access, and programmable input-output. SCALE-MAMBA uses a multi-threaded architecture for the precomputation phase.

Support for SCALE-MAMBA development was discontinued in 2023 [137]. FANNG-MPC (Section 2.4.14) is a fork of SCALE-MAMBA seeking to create a new MPC framework which is used, e.g. by CipherCompute (Section 2.4.4).

2.4.36 SEAL

Microsoft Simple Encrypted Arithmetic Library (SEAL)[67] offers implementations of HE schemes and provides an API for writing protocols. It supports addition and multiplication of encrypted real numbers and integers. It supports BFV, BGV and CKKS schemes.

2.4.37 Secrecy

Secrecy [158][68] is a passively secure computation framework based on a three-party replicated secret sharing protocol [70]. The focus of the project is on optimising SQL queries for an MPC backend. Work has been done towards supporting the MP-SPDZ (Section 2.4.28) backend in Se-

[62]PySyft `https://github.com/OpenMined/PySyft` Last accessed: October 2024
[63]SyMPC `https://github.com/OpenMined/SyMPC/` Last accessed: October 2024
[64]Sycret `https://github.com/OpenMined/sycret` Last accessed: October 2024
[65]UN Statistics wiki. *United Nations PET Lab: International Trade* `https://unstats.un.org/wiki/display/UGTTOPPT/16.+United+Nations+PET+Lab%3A+International+Trade` Last accessed: October 2024
[66]SCALE-MAMBA `https://github.com/KULeuven-COSIC/SCALE-MAMBA` Last accessed: October 2024
[67]SEAL `https://github.com/microsoft/SEAL` Last accessed: October 2024
[68]Secrecy `https://github.com/CASP-Systems-BU/Secrecy` Last accessed: October 2024

crecy, as well as using Carbyne Stack[69] (Section 2.4.2). Secrecy is developed by Boston University. It has been used for a digital health study and secure cross-site analytics[70].

2.4.38 SecretFlow

SecretFlow [159, 160][71] is a framework for privacy-preserving data analysis and machine learning. It has backends using secure multiparty computation, trusted execution environments, and homomorphic encryption. It also supports differential privacy. Its Secure Processing Unit has a Python API for ML application development; this is compiled to PPHLO (Privacy-Preserving High-Level Operations) executed by the backend. The MPC backends include a passively secure variation of ABY^3 for three parties protocols, SPD$\mathbb{Z}_{2^k}$ for n-parties, and Cheetah [161] for two parties. It also includes a Secure Collaborative Query Language (SCQL) – a system that translates SQL to mixed public and MPC computations. SecretFlow also has a homomorphic encryption processing unit. SecretFlow is developed by Ant Group Co.

2.4.39 SEEC

SEEC [162][72] is a framework for memory safety in passively secure two-party computation. It uses the GMW protocol and the Boolean part of ABY (Section 2.4.27). It is developed by the Cryptography and Privacy Engineering Group at TU Darmstadt.

2.4.40 Sequre

Sequre [163][73] is a recent framework proposed for bioinformatics pipelines. It is based on additive secret sharing and homomorphic encryption and offers passive security. It operates in a trusted dealer model where the dealer generates the randomness the online phase needs from the precomputation phase. It proposes optimisations to the specific cases of the Beaver multiplication protocol where the values can be cached and other specialised protocols.

2.4.41 Senate

Senate [164] is an actively secure MPC framework for analytics and SQL queries. The idea is to divide computations to subcircuits so that only a subset of parties evaluates each circuit. The secure evaluation uses authenticated multiparty garbled circuits based on [106] implementation in EMP (Section 2.4.10).

[69]Bosch Tube: John Liagouris. *Secrecy: Secure collaborative analytics in untrusted clouds* 2022 https://bosch-ext.mediaspace.de.kaltura.com/media/SecrecyA+Secure+collaborative+analytics+in+untrusted+clouds/0_j66zkevf Last accessed: October 2024

[70]Bosch Tube: John Liagouris. *Secrecy: Secure collaborative analytics in untrusted clouds* 2022 https://bosch-ext.mediaspace.de.kaltura.com/media/SecrecyA+Secure+collaborative+analytics+in+untrusted+clouds/0_j66zkevf Secrecy is used in https://www.bu.edu/hic/research/focused-research-programs/continuous-analysis-of-mobile-health-data-among-medically-vulnerable-populations/ and https://research.redhat.com/blog/research_project/secure-cross-site-analytics-on-openshift-logs/ Last accessed: October 2024

[71]SecretFlow https://www.secretflow.org.cn/ source code in https://github.com/secretflow Last accessed: October 2024

[72]SEEC https://github.com/encryptogroup/SEEC/ Last accessed: October 2024

[73]Sequre https://github.com/0xTCG/sequre Last accessed: October 2024

2.4.42 SEPIA

SEPIA [165][74] (SEcurity through Private Information Aggregation), was one of the earlier MPC frameworks and used Shamir's secret sharing. It is secure in a passive model with an honest majority. SEPIA specified an API for programmers but did not include its own programming language. It is no longer being maintained. An extension of SEPIA with an offline phase [166] was developed prior to this approach becoming widely used.

2.4.43 Sharemind MPC

Sharemind MPC framework[75] is a secure multiparty computation platform based mainly on additive secret sharing for both active and passive security. There are prototypes using garbled circuits and homomorphic encryption as well as support for ABY and FRESCO backends. Sharemind MPC is maintained by Cybernetica as a commercially available product with documentation for application developers being public [76] and tools published as open source[77].

While original versions of Sharemind MPC were limited to a specific three-party protocol set, the current release of Sharemind MPC is compatible with multiple secure computing approaches with varying number of computing parties and security guarantees. An incomplete list is provided in [167]. The most efficient and further developed protocol suite in the Sharemind MPC framework is the three-party passively secure protocol based on additive secret sharing over $\mathbb{Z}_{2^k}$ that is secure against one corrupted party [168]. It supports integer, fixed-, and floating-point data, as well as comparison operations [168], bit-shifts, real numbers [169, 170]), AES block cipher invocation [171], shuffling and sorting [172], and oblivious RAM [173].

Some Sharemind MPC protocol implementations are developed using a protocol description language (PDSL) [174]. The PDSL optimises the protocols and generates code that is tens of times more efficient than hand-implemented code [175]. The privacy of protocols implemented using the PDSL can also be proved automatically using a special tool [34]. SecreC [175], a C-like high-level language, is used for specifying private computation tasks. Sharemind MPC comes with a standard library of operations for descriptive statistics, quality control, outlier removal, statistical testing, regression analysis, and more. The SecreC programming language reference[78] and the SecreC standard library reference[79] are publicly available online. One potential frontend for Sharemind MPC is the R-like statistical analysis engine called Rmind [176] which allows data scientist to explore and analyse a database without being able to see individual records.

[74] SEPIA `https://sites.google.com/view/sepia-mpc/startseite` Last accessed: October 2024

[75] Sharemind MPC. Cybernetica. `https://cyber.ee/products/sharemind-mpc` Last accessed: October 2024.

[76] Sharemind Developer Zone `https://docs.sharemind.cyber.ee/sharemind-mpc` Last accessed: October 2024

[77] Sharemind software development kit `https://github.com/sharemind-sdk` Last accessed: October 2024

[78] SecreC language reference. `https://docs.sharemind.cyber.ee/sharemind-mpc/2023.09/development/secrec-reference.html` Last accessed: October 2024

[79] SecreC Standard Library. `https://docs.sharemind.cyber.ee/2023.09/api/secrec-stdlib` Last accessed: October 2024

Sharemind MPC has been used for statistical studies in Estonia [177, 178], as well as by Asemio in an antipoverty and early childhood education study in USA[80], and in a multi-site clinical study between hospitals in Germany and Italy[81] [179].

2.4.44 Silent Compute

Silent Compute[82] is a framework for privacy-preserving data analytics. According to their webpage, they support basic arithmetic (addition, subtraction, multiplication, division or a function with a combination of these), querying function (sorting, joining, filtering), and machine learning based on Shamir's secret sharing. Silent Compute is developed by Silence Laboratories.

2.4.45 Swanky

Swanky[83] is a suite of Rust libraries for secure computation. Swanky currently contains passively secure garbled circuits for both Boolean and arithmetic case, protocols for secure set intersection, and tools for zero-knowledge proofs. It aims to cover active security in the future. Swanky is developed by Galois Inc and has been used for sharing Department of Education data in the US [180].

2.4.46 Symphony

Symphony [181][84] is an MPC programming language inspired by Wysteria (Section 2.4.54), EMP (Section 2.4.10) and Obliv-C (Section 2.4.30). Its goal is to combine the best ideas of the previous languages. Uses EMP (Section 2.4.10) and MOTION (Section 2.4.27) as MPC backends. It is developed by the Programming Languages Group at University of Maryland.

2.4.47 Tandem

Tandem[85] is a framework for actively secure two-party computation using garbled circuits based on [103]. Applications can be programmed in Garble[86], a language for garbled circuits programming. Garble is statically typed, low-level, purely functional and inspired by Rust. Tandem is developed by the SINE foundation – a think-and-do tank offering solutions for data sharing dilemmas.

[80]Asemio. *How Tulsa Is Preserving Privacy and Sharing Data for Social Good* 2019 `https://asemio.com/wp-content/uploads/2022/10/How-Tulsa-is-Preserving-Privacy-and-Sharing-Data-for-Social-Good-2.pdf` Last accessed: October 2024

[81]LMU Munich. *Data security: Breakthrough in research with personalized health data*, 2024. `https://www.lmu.de/en/newsroom/news-overview/news/data-security-breakthrough-in-research-with-personalized-health-data.html` Last accessed: October 2024

[82]Silent Compute `https://www.silencelaboratories.com/silent-compute` Last accessed: October 2024

[83]Swanky `https://github.com/GaloisInc/swanky` Last accessed: October 2024

[84]Symphony `https://github.com/plum-umd/symphony-lang` Last accessed: October 2024

[85]Tandem `https://github.com/sine-fdn/tandem` Last accessed: October 2024

[86]Garble `https://github.com/sine-fdn/garble-lang` Last accessed: October 2024

2.4.48 TASTY

TASTY [121] (Tool for Automating efficient Secure Two-partY)[87] is a framework combining garbled circuits and homomorphic encryption for secure two-party computation. It was the first framework to compile circuits for its own use. It is no longer being maintained.

2.4.49 TF-Encrypted

TF-Encrypted[88] is a framework for privacy-preserving deep learning in TensorFlow. It has a secure multiparty computation backend, Moose[89], that uses three-party replicated secret sharing in the passive security model based on [70]. Moose supports integer and fixed-point arithmetic. It also supports a version of the SPDZ protocol called Pond[90] where precomputation is done by a trusted party, SecureNN [182], and ABY3[91]. TF-Encrypted was created by Cape Privacy (when it was known as Dropout Labs) and is currently a community project maintained by TF-Encrypted[92].

2.4.50 TinyGarble

TinyGarble [183][93] uses hardware circuit generation tools to optimise circuits for garbled circuits. It also implements a garbled circuits protocol with several optimisations. This work is succeeded by TinyGarble 2.0 [51][94] for garbled circuit evaluation with both passive and active security that uses some components of the EMP toolkit (Section 2.4.10). It covers several optimisations as well as abstractions to ease the use of garbled circuits.

2.4.51 TNO-MPC

TNO-MPC[95] is a collection of tools and protocols for MPC. It includes, inter alia, implementations of Shamir's secret sharing and Paillier encryption. It also uses the MPyC framework for some computations. TNO-MPC implements protocols for secure comparison and approximate matching as well as inner joins using Paillier encryption. These tools are developed by the Netherlands Organisation for Applied Scientific Research (TNO) PET Lab. TNO has developed privacy-preserving financial risk score propagation [184]. TNO has also developed an MPC system to identify heart failure risks[96]; however, it is unclear if the tools in their open source library were used.

[87]TASTY `https://github.com/encryptogroup/tasty` Last accessed: October 2024
[88]TF-Encrypted source code `https://github.com/tf-encrypted` Last accessed: October 2024
[89]Moose `https://github.com/tf-encrypted/moose` Last accessed: October 2024
[90]Ben DeCoste. *Announcing SecureNN in tf-encrypted* 2018 `https://medium.com/dropoutlabs/announcing-securenn-in-tf-encrypted-9c9c3e8a5a52` Last accessed: October 2024
[91]TF-Encrypted protocol `https://github.com/tf-encrypted/tf-encrypted/tree/master/tf_encrypted/protocol` Last accessed: October 2024
[92]TF-Encrypted `https://tf-encrypted.io/` Last accessed: October 2024
[93]TinyGarble `https://github.com/esonghori/TinyGarble` Last accessed: October 2024
[94]TinyGarble2.0 `https://github.com/IntelLabs/TinyGarble2.0` Last accessed: October 2024
[95]TNO-MPC `https://github.com/TNO-MPC/` Last accessed: October 2024
[96]Marie Beth van Egmond. *Identifying heart failure patients at high risk using MPC* 2020 `https://medium.com/applied-mpc/identifying-heart-failure-patients-at-high-risk-using-mpc-ab8900e75295` Last accessed: October 2024

2.4.52 VIFF

Virtual Ideal Function Framework (VIFF)[185][97] was the first framework for asynchronous protocols. For asynchronous behaviour, it also featured a scheduler that allowed the evaluation of operations as soon as the inputs were available. The protocol set offers security for active adaptive corruption of up to $n/3$ parties using Shamir's secret sharing in the BGW protocol. The protocol does not guarantee the success of the preprocessing, but in case of successful preprocessing, the online protocol has guaranteed output delivery for the honest parties. Implementation supports 3 or more parties and 32-bit numbers. Protocols for VIFF can be written in Python. VIFF is no longer being developed.

2.4.53 Virtual Data Lake

Virtual Data Lake (VDL)[98] offers an MPC engine and a Python package to interact with it. Their MPC is based on the BGW protocol using Shamir's secret sharing and offers semi-honest security for honest majority[99]. VDL supports signed and unsigned integers, Booleans, fixed-point numbers and strings. Users can use a Python package called crandas[100] to get access to data structures and analysis tools. The crandas package is designed to be similar to the pandas Python package but for use with the VDL secure computation backend. Operations supported by crandas include filtering, shuffling, slicing, merging and joining tables, linear and binomial logistic regression for statistics. VDL is developed by Roseman Labs and has been used to collect information about digital threats[101].

2.4.54 Wysteria

Wysteria [186][102] is a high-level functional language with an interpreter running a Boolean-circuit version of the GMW protocol. Wysteria also automatically verifies that the compiled secure computation protocol indeed has the same functionality as the program implemented by the developer in the high level language. A verified extension of it called Wys* is considered in [187].

2.4.55 XOR

Inpher's XOR[103] is a service by Inpher with a FHE-based backend using TFHE scheme and MPC backend with Manticore (Section 2.4.26) and a trusted dealer. XOR is mostly directed at privacy-preserving machine learning. XOR has been used for financial fraud detection[104] and financial intelligence sharing[105]. It is also an enabling component in an Analytics Network for Asset Managers with CPP Investments and privacy-preserving data collection environment with DataCo.

[97] VIFF https://github.com/mgeisler/viff Last accessed: October 2024

[98] Virtual Data Lake by Roseman Labs https://rosemanlabs.com/ Last accessed: October 2024

[99] Roseman Labs blog. *Beyond Apple's Homomorphic Encryption software* 2024 https://rosemanlabs.com/en/blogs/beyond-apples-homomorphic-encryption-software Last accessed: October 2024

[100] Roseman Labs blog. *About Roseman Labs' package, crandas* 2024 https://rosemanlabs.com/en/blogs/about-roseman-labs-package-crandas Last accessed: October 2024

[101] Roseman Labs blog. *SecureNed, important building block for Public-Private information sharing* 2024 https://rosemanlabs.com/en/blogs/securened-important-building-block-for-public-private-information-sharing Last accessed: October 2024

[102] Wysteria https://bitbucket.org/aseemr/wysteria/wiki/Home Last accessed: October 2024

[103] XOR secret computing https://inpher.io/xor-secret-computing/ Last accessed: October 2024

[104] https://inpher.io/solutions/by-industry/financial-services/ Last accessed: October 2024

[105] The Global Coalition to Fight Financial Crime. *Future of Financial Intelligence Sharing (FFIS) Innovation and discussion paper: Case studies of the use of privacy preserving analysis to tackle financial crime* 2020

2.4.56 XSCE

XSCE (XDP Secure Computing Engine)[106] is an MPC framework in the XDP platform based on MP-SPDZ (Section 2.4.28). XSCE combines secret sharing, homomorphic encryption, and trusted execution environments. XDP is developed by ParityBit Technologies.

2.5 Overview and Comparison of the Secure Multiparty Computation Tools

The goal of this section is to summarise the information from Section 2.4. First, Section 2.5.1 references some literature that focuses on comparing the listed tools. Section 2.5.2 groups the tools included in this overview according to their main features. Some of the previously mentioned tools are frameworks that enable secure computation while others are tools that support the development but do not offer full solutions.

2.5.1 Prior Comparisons

Various prior works have attempted to compare some of the secure computation frameworks and tools. In addition, most MPC framework publications also present their own comparison to prior works. This section lists some papers that have focused on comparing different secure computation platforms. Notably, it is the goal of MP-SPDZ [150] to enable efficient benchmarking of different proposed protocols. Also, [73] gives a good overview of protocols with dishonest majority in the active security setting.

A comparison of EMP, Obliv-C, ObliVM, TinyGarble, Wysteria, ABY, SCALE-MAMBA, Sharemind MPC, PICCO, Frigate and GBMC-GC can be found in [188]. This comparison focuses on general purpose compilers providing a language to describe the functions and executing the secure computation protocols underneath. In addition to functionality and features of the MPC frameworks, this work also considers usability as an aspect for comparison. More specifically, it focuses on the expressability of the language used to program MPC applications, core secure computation method, number of computing parties, security model, level of documentation and available support. Follow-up work is presented at `https://github.com/MPC-SoK/frameworks`. Previously, the Bar Ilan University also used Libscapi to implement comparable versions of different MPC protocols[107]. Another tool for MPC benchmarking is also being developed by TU Darmstadt[108]. Currently it contains ABY, MOTION, MP-SPDZ and SEEC but no overview of the findings has been published.

MP-SPDZ and MPyC are compared and profiled in [189, 190]. Both works benchmark the execution of basic operations like addition and multiplication and report that MP-SPDZ outperforms MPyC and has a more optimised compiler. The comparison in [189] notes that the asynchronous architecture of MPyC can outperform MP-SPDZ for high latency networks.

Lattigo, SEAL, PALISADE, HEAAN, and other tools for fully homomorphic encryption are compared in [191, 192]. The goal of [192] is to propose benchmarks suitable to evaluate the performance of the homomorphic computation while [191] focuses on benchmarking specific applications. Both works compare the implemented encryption algorithms and the language of implementation.

`https://www.gcffc.org/wp-content/uploads/2020/06/FFIS-Innovation-and-discussion-paper-Case-studies-of-the-use-of-privacy-preserving-analysis.pdf` Last accessed: October 2024

[106]XSCE `https://github.com/paritybit-ai/XSCE` Last accessed: October 2024

[107]MPC Benchmarking tool `https://github.com/cryptobiu/MPC-Benchmark` Last accessed: October 2024

[108]MPC benchmarking tool `https://github.com/encryptogroup/mpc-bench` Last accessed: October 2024

[191] also provides an account of the accessibility of the source code and documentation and their last updates.

An overview of secure computation techniques applied for machine learning can be found in [193].

The correctness of MPC compilers in MP-SPDZ, EzPC and EMP is analysed in [194] which detected many errors that either crashed the compilers or created incorrect MPC protocols. This work also proposes a testing framework for such compilers that can be used by follow-up work to ensure better quality of MPC compilers. A similar line of work for the verification of correctness of deep learning with MPC with PySyft, TF-Encrypted and CrypTen is carried out in [195]. They also managed to find several cases where the outputs of the MPC version and public version of the algorithms deviated.

2.5.2 Classification

The list presented in Section 2.4 covered tools for various aspects of secure multiparty computation and its use. Many of the tools share common characteristics in terms of functionality or purpose and some multi-purpose tools cover several aspects at once. The tools listed here can be classified as those providing compilation or language support for MPC, libraries to develop MPC, frameworks that facilitate running MPC, and platforms for large scale MPC deployments as follows.

2.5.2.1 Compilers and Languages

First, a number of compilers and languages exist that enable users to program applications for secure multiparty computation. Some tools generate circuits from a higher level language: CBMC-GC, CirC, Frigate, OblivC, ObliVM, Senate. There are also languages like Symphony, Wysteria, and Secrecy that can support various MPC backends. Senate can be seen either as a framework or an interface making the EMP toolkit more easy to use for data analytics. Similarly, the focus of Secrecy is on optimising SQL queries for its MPC backend. Sequre is also a framework, but the focus is on detecting, which parts of a bioinformatics pipeline need to be executed using secure computation.

HyCC is a compiler for a mixed-mode MPC protocol enabling to efficiently choose which protocol to execute. Similar functionality is also supported by the EzPC and SecretFlow frameworks. FUSE and COMBINE are both recently proposed intermediate representations that enable various optimisations of the circuits for secure computation and support mixed-mode protocols. They both support MP-SPDZ and MOTION as secure computation backends.

Note that many of the frameworks like EMP, PICCO, Sharemind MPC, and Tandem also define their own languages. For Sharemind MPC, the SecreC language can be used to develop applications and a PDSL exists for protocol optimisation. While SecreC currently targets Sharemind MPC, it is open source and support for other MPC backends can be added with a significant amount of its standard library functionality automatically being ported over. Some other frameworks like MP-SPDZ, MPyC, SCALE-MAMBA, and Virtual Data Lakes support languages that closely follow common programming languages like Python, Rust, or C. However, this should not be taken to mean that one can execute any program in these languages with these MPC frameworks. Rather, they define languages like the core language or specific packages for these languages that can then be used to develop code for the MPC applications.

2.5.2.2 Libraries and APIs

Some tools in Section 2.4 are providing libraries or APIs to enable programming secure multiparty computation applications. There are various libraries for developing homomorphic encryption, like Lattigo, Libscapi, OpenFHE, SEAL. These can be and have been used to develop secure multiparty computation applications. TNO-MPC libraries include both homomorphic encryption as well as secret sharing for secure computation. Swanky and JIFF are intended for developing secure multiparty computation.

2.5.2.3 MPC Frameworks

Most of the tools listed in Section 2.4 are general MPC frameworks. In this context, the term *framework* refers to a collection of tools to facilitate secure computations by performing orderly execution of MPC protocols. Hence a framework exhibits some form of a runtime system and interfaces for running user-defined MPC applications. Frameworks vary in terms of features: some have extra capabilities necessary for real deployments, e.g. I/O and networking, while others serve as minimal proofs of concept or academic implementations.

A summary of MPC frameworks is given in Table 2. The **Method** column indicates the core secure computation method, with FHE standing for fully homomorphic encryption, GC for garbled circuits, SS for secret sharing, and 'mixed' for a combination of the protocols. Mixed-mode protocols are often a combination of SS and GC, and occasionally also FHE, especially in the precomputation phase. Uses of TEE and FSS are explicitly noted in the table as these are emerging trends in MPC likely to grow further in the near future. **# of Parties** indicates the exact number of computing parties involved in the secure computation or the lower limit thereof. In the latter case, a '$+$' sign is added, e.g. $2+$ means for 2 or more parties. For protocols requiring a helper party, this helper party is counted as one additional party. For open source solutions, the table also lists the year of the last update in their repositories. The **Application Development** column lists programming languages and other supporting tools that users may use to build applications for the framework. For the cases where the table lists Python, Rust or C, the main idea is that the respective language is similar to these languages or represents a special version thereof. The column for real-life applications indicates if there are known applications with real data for this framework. All these applications are referenced in the respective subsections in Section 2.4. Empty cells in the table indicate that no information was found.

Of the listed frameworks, Fairplay, SEPIA, TASTY, TinyGarble, and VIFF are mainly of historical importance. SCALE-MAMBA is also no longer being maintained but lives on in FANNG-MPC and CipherCompute and has been actively used in research. On the other side, frameworks like Asterisk, FANNG-MPC, Helium, SEEC, Sequre, SecretFlow are very recent proposals and it remains to be seen if they will eventually develop beyond the stage of academic implementations supporting merely a short string of scientific publications. Some frameworks, such as EMP, MP-SPDZ, EzPC, MOTION, MPyC have already been well established in the academic setting, are actively developed, and are important research tools. Others, such as CipherCompute, Demeter, SecretFlow, Sharemind MPC, Virtual Data Lake, and XOR also have commercial offerings.

There are many recent applications for secure computation in privacy-preserving machine learning. Some design their own protocols or implement their own versions of existing protocols while others use existing frameworks like MP-SPDZ and MOTION. Frameworks like FANNG-MPC, EzPC, PySyft, SecretFlow, XOR, TF-Encrypted, and versions of MOTION are implemented with privacy-preserving machine learning as the main target application but they can also support other kinds of computation.

Many others, such as CipherCompute, FRESCO, Helium, Manticore, MOTION, MPyC, MP-SPDZ, SCALE-MAMBA, Sharemind MPC, Tandem, Virtual Data Lake and XSCE are intended for generic secure computation tasks. Sharemind MPC, Silent Compute, and Virtual Data Lakes are especially tailored for data analysis applications. Divvi Up also supports specific aggregated statistics computations. Some frameworks, like HybrTC, Secrecy and Senate are focused on supporting database operations.

For the JOCONDE System, the focus is on three-party secure computation in the active security model with a dishonest majority. From Table 2, the existing MPC frameworks that could fulfil these requirements are CipherCompute, EMP, EzPC, FANNG-MPC, FRESCO, FudanMPL, MP-SPDZ, PySyft, SCALE-MAMBA, Senate, Sharemind MPC, TF-Encrypted, and XSCE. Furthermore, as discussed in Chapter 4, passive security MPC frameworks can be considered in combination with TEE.

Table 2. Summary of MPC frameworks

Framework	Security Model	Method	# of Parties	Open Source	Last Up-dated	Application Development	Focus	Real-Life Applications
Asterisk	active dishonest majority with helper party	SS	$3+$	yes	2024		Large scale secure computation	
CipherCompute	active dishonest majority	mixed	$2+$	yes	2021	Rust	Generic secure computation	
Demeter				no			Blockchain	
Divvi Up	active dishonest majority for privacy, passive for correctness	SS	$2+$	yes	2024		Aggregate statistics, collecting telemetry data	yes
EMP	passive and active dishonest majority	GC	$2+$	yes	2024	Symphony, Senate	Garbled circuits research	yes
EasySMPC	passive dishonest majority	SS	$2+$	yes	2023	Graphical user interface	Ease of MPC development, biomedical data sharing	
EzPC	passive and active	mixed, FSS, TEE	$2, 3$	yes	2024	C-like	Machine learning	
Fairplay	passive	GC	$2+$	yes	2015	SFDL, CBMC-GC	Generic secure computation	
FANNG-MPC	active dishonest majority	mixed	$2+$	yes	2024	Rust	Machine learning	
FBPCP		mixed	2	yes	2024		Privacy-preserving advertising	yes
FRESCO	passive and active dishonest majority	SS	$2+$	yes	2024		Generic secure computation	yes
FudanMPL	passive and active	mixed	$2+$	yes	2024		Machine learning	

Table 2. Summary of MPC frameworks *(continued)*

Framework	Security Model	Method	# of Parties	Open Source	Last Updated	Application Development	Focus	Real-Life Applications
Helium	passive	FHE	$2+$	yes	2024		Generic secure computation	
HybrTC	active computing parties, passive input and output parties	mixed, TEE	$2+$	no			Database queries	
JIFF	passive	SS	$2+$	yes	2024	Javascript	Web applications	yes
Manticore	passive dishonest majority	mixed	$2+$	no			Generic secure computation	
MOTION	passive dishonest majority	mixed	$2+$	yes	2023	COMBINE, FUSE, HyCC	Generic secure computation	
MP-SPDZ	passive and active	mixed	$2+$	yes	2024	Python, COMBINE, FUSE	MPC protocol benchmarking	
MPyC	passive honest majority	SS	$3+$	yes	2024	Python	Generic secure computation	yes
PICCO	passive honest majority	SS	$3+$	yes	2024	C	Compiler from C to MPC	
PySyft	active and passive	mixed, FSS	$2+$	yes	2024	Python	Machine learning	yes
SCALE-MAMBA	active dishonest majority	mixed	$2+$	yes	2022	Rust	Generic secure computation	
Secrecy	passive	SS	3	yes	2023	SQL	Database operations	yes
SecretFlow	passive	mixed, TEE	$2+$	yes	2024	Python	Data analysis and machine learning	
SEEC	passive	mixed	2	yes	2024		Memory safe implementation	

Table 2. Summary of MPC frameworks *(continued)*

Framework	Security Model	Method	# of Parties	Open Source	Last Updated	Application Development	Focus	Real-Life Applications
Sequre	passive trusted dealer	mixed	$3+$	yes	2024	Python	Biomedical data sharing	
Senate	active	GC	$2+$	no		SQL	Database operations	
SEPIA	passive honest majority	SS	$3+$	yes	2018	API, no language	Generic secure computation	
Sharemind MPC	passive and active	mixed	$2, 3$	partial	2023	SecreC, Rmind	Generic secure computation	yes
Silent Compute		SS	$2+$	no			Data analysis	
Tandem	active	GC	2	yes	2024	Garble	Generic secure computation	
TASTY	passive	mixed	2	yes	2016		Generic secure computation	
TF-Encrypted	passive and active	SS	3	yes	2024		Machine learning	
TinyGarble	passive and active	GC	2	yes	2023		Generic secure computation	
VIFF	active honest majority	SS	$3+$	yes	2014	Python	Generic secure computation	
Virtual Data Lake	passive honest majority	SS	$3+$	no		Python, crandas	Data analysis	yes
XOR	passive	mixed	$2+$	no			Machine learning	yes
XSCE	active	mixed, TEE	$2+$	yes	2024	Python	Generic secure computation	

2.5.2.4 MPC Platforms

An MPC platform can be thought of as an extension to an MPC framework with additional consideration for practical deployments, e.g. workflows and automation. Distinctly, platforms do not represent ad-hoc MPC deployments but should be able to serve a wide range of MPC applications in a single deployment.

Carbyne Stack is an example of a dedicated platform. Employing MP-SPDZ as its MPC framework, it provisions supplementary services for persistent storage, precomputation, and client APIs. Similarly, Sharemind MPC offers platform functionality with its Application Server for orchestrating task executions, client authentication and authorisation, persistent storage, and configuration. EasySMPC, SEPIA, and Divvi Up exhibit rudimentary platform features, enabling clients to outsource use case-specific MPC tasks to a set of computing parties with user-defined input data. Virtual Data Lake, XOR, and CipherCompute are solutions built around proprietary general-purpose MPC platforms. Being catered for enterprise use cases, these solutions are closer to what can be considered as an MPC service.

MPC service solutions. There is no single interpretation of what an MPC service (or MPC-as-a-Service) is. One way is to see it as a *marketplace*, connecting prospective MPC users with a set of computing nodes. In the Partisia Blockchain Infrastructure, an MPC-as-a-Service facilitates acquiring a group of MPC nodes from a node pool to allocate a certain task; the allocated node group in combination with a pre-defined task forms the MPC service[109]. This approach aims to establish a self-sustained ecosystem of buyers and service providers.

In an alternative view, the MPC service is seen as the instance of a platform serving a dynamic environment for enrolled users to organise and execute MPC tasks on demand. Examples of this include the XOR Service[110] which remotely orchestrates the execution of tasks, and Virtual Data Lake[111], in which the MPC nodes are operated by the service provider. Finally, the term MPC-as-a-Service can be used to characterise a system that focuses on end-user serviceability, offering a comprehensive abstraction for an MPC platform that is reminiscent of traditional information systems. Both Roseman Labs and CipherCompute offer a self-service web interface for users to organise and deploy tasks without external help: specify the computation, assign roles, approve computation details, upload input data and download output data.

The MPC-as-a-service system of JOCONDE is, in essence, closer to the latter model. However, contrary to the given examples, in which a platform is integral to the service, the JOCONDE System is envisioned as an independent layer of abstraction on top of a selected set of existing MPC platforms.

2.6 Published MPC Projects Using Large Scale Real Data

A list of MPC deployments is managed by UC Berkeley[112] and a list of applications using privacy-enhancing technologies, including MPC and HE, is hosted by Centre for Data Ethics and Inno-

[109] Partisia Blockchain. *MPC Techniques Series, Part 10: MPC-as-a-Service — the Partisia Blockchain Infrastructure* https://medium.com/partisia-blockchain/mpc-techniques-series-part-10-mpc-as-a-service-the-partisia-blockchain-infrastructure-9b4833e77965 Last accessed: October 2024

[110] XOR Service https://dev.inpher.io/xor/concepts/#xor-service Last accessed: October 2024

[111] Roseman Labs. *Roseman Labs Help Center - I want to understand the product better* https://support.rosemanlabs.com/i-want-to-understand-the-product-better Last accessed: October 2024

[112] A hub for real-life MPC deployments https://mpc.cs.berkeley.edu/ Last accessed: October 2024

vation[113]. The United Nations guide on privacy-enhancing technologies [196] also lists several of the secure computation frameworks as tools to be considered for official statistics and gives examples of real-life deployments. Some examples of MPC used in various situations are also listed in [197] by the Centre of Excellence for Data Sharing and Cloud (CoE-DSC). Several companies operating in the MPC space can be found in the list of members of the MPC Alliance[114].

The majority of published MPC use cases focus on proving the technical feasibility of the approach and benchmarking the computation results. For these purposes, they mostly use synthetic data or real-world data that are already publicly available, rather than real-world confidential data.

The deployment of secure multiparty computation with real-world confidential data requires overcoming organisational and legal obstacles on top of technical ones. Being used with large-scale real-world confidential data is an indication of maturity of the MPC platform and experience of the involved company in dealing with various MPC deployment aspects. Below, we have listed MPC projects that have managed to process real-world confidential data.

Bold text indicates the **framework**. The domain or purpose of the project is presented together with the customer organisation and the country where the project was conducted. If available, the computing parties involved in the project are included.

Virtual Public Register (VPR) by Partisia with MPC backend in **FRESCO** is used in the OSCAR project[115] and its ongoing extension OSCAR Dream[116] in Denmark. The computing parties are the Danish Health Data Organisation (SDS) and Statistics Denmark (DST)[117]. A series of preparatory projects led up to the OSCAR project[118].

Virtual Data Lake was used by University Medical Center Utrecht and Roseman Labs in The Netherlands for studying the effectiveness and value of podiatric care[119].

Virtual Data Lake is used by the Municipality of Rotterdam and Roseman Labs in The Netherlands for empowering disadvantaged toddlers for educational success[120].

Virtual Data Lake is used by the Municipality of Rotterdam and Roseman Labs in The Netherlands to study adverse childhood experiences[121]. This project is in development and should be released in 2025.

[113]Centre for Data Ethics and Innovation. *Repository of Use Cases* https://cdeiuk.github.io/pets-adoption-guide/repository/ Last accessed: October 2024

[114]MPC Alliance https://www.mpcalliance.org/ Last accessed: October 2024

[115]The OSCAR project https://www.oscar-project.com/ Last accessed: October 2024

[116]The OSCAR Dream project https://www.oscar-project.com/oscar-dream Last accessed: October 2024

[117]Virtual Public Register (VPR) Platform by Partisia https://www.partisia.com/products/confidential-computing/ Last accessed: October 2024

[118]Series of project leading to OSCAR project https://onkologisktidsskrift.dk/samfund/1895-oscar-projektet-skal-sikre-storre-indsigt-i-sundhedsudfordringer.html Last accessed: October 2024

[119]Roseman Labs. *Dataspace Officially Launched: Breakthrough In (Preventive) Foot Care. First large-scale study of the results of podiatric care in diabetes mellitus.* https://rosemanlabs.com/en/customers/nederlandse-vereniging-van-podotherapeuten-nvvp Last accessed: October 2024

[120]Roseman Labs. *Gemeente Rotterdam empowers disadvantaged toddlers for educational success* https://rosemanlabs.com/en/customers/gemeente-rotterdam-empowers-disadvantaged-toddlers-for-educational-success Last accessed: October 2024

[121]Roseman Labs. *UMC Utrecht: When patient data is too sensitive to share* https://rosemanlabs.com/en/customers/umc-utrecht-when-patient-data-is-too-sensitive-to-share Last accessed: October 2024

TNO (now its spinoff Linksight) worked with The Netherlands health insurer CZ, the Limburg hospital Zuyderland Medical Center, and the Central Bureau of Statistics 2021 to analyse insurance and medical outcome data [122].

The Linksight platform is used for monitoring elderly care in The Netherlands (Delft, Schieland, Westland regions).[123].

A **Paillier encryption scheme** implementation with threshold decryption is used for anti-money laundering [198] in collaboration by TNO, ABN AMRO, and Rabobank in The Netherlands. Feasibility of the approach is estimated on real data from a Dutch bank. Benchmarking is done on synthetic data.

XOR is used for business intelligence and marketing for a financial product by bank subsidiaries in The Netherlands, Belgium, and Luxembourg[124].

Sharemind MPC was used in a privacy-preserving analysis of tumor treatment radiotherapy data from two separate hospitals: the LMU University Hospital in Munich and the Policlinico Universitario Fondazione Agostino Gemelli in Rome[125] [179].

Sharemind MPC was used by the Tartu City Government's Human Resources Department for an employee satisfaction survey of Tartu city government employees[126] [199]. Computing parties were Alexandra Institute (Denmark), Cybernetica (Estonia) and Partisia (Denmark).

Sharemind MPC was used by CentAR (Estonia) to analyse data of students with special education needs. The computing parties involved were the Ministry of Education, Ministry of Finance Information Technology center RMIT, and Cybernetica, all in Estonia. The public cloud service provider Zone.ee was used to host one of the computation servers. The other servers were hosted in the private clouds of the computing parties[127].

Sharemind MPC was used by Asemio in USA to study gaps in anti-poverty and early childhood education service delivery in order to find children in risk families for educational programmes[128].

[122]TNO Linksight. *Privacy-preserving analysis of patient data* 2022 https://www.linksight.nl/en/content/ppa/ Last accessed: October 2024

[123]Linksight. *Governance design for MPC data collaboration: Case study on the data collaboration in elderly care sector monitoring impact of policies and measures taken on the state of care* 2023 https://coe-dsc.nl/wp-content/uploads/2023/06/governance-for-an-mpc-data-collaboration-based-on-elderly-care-monitoring-case.pdf Last accessed: October 2024

[124]The Global Coalition to Fight Financial Crime. *Future of Financial Intelligence Sharing (FFIS) Innovation and discussion paper: Case studies of the use of privacy preserving analysis to tackle financial crime* 2020 https://www.gcffc.org/wp-content/uploads/2020/06/FFIS-Innovation-and-discussion-paper-Case-studies-of-the-use-of-privacy-preserving-analysis.pdf Last accessed: October 2024

[125]LMU Munich. *Data security: Breakthrough in research with personalized health data*, 2024. https://www.lmu.de/en/newsroom/news-overview/news/data-security-breakthrough-in-research-with-personalized-health-data.html Last accessed: October 2024

[126]PRACTICE project. *Pilot of the secure survey system created in PRACTICE* 2016 https://practice-project.technikon.com/blog/entry/pilot-of-the-secure-survey-system-created-in-practice.html. Last accessed: October 2024

[127]Cybernetica. *National Special Education Data Analysed Securely* 2017 https://cyber.ee/resources/news/national-special-education-data-analysed-securely Last accessed: October 2024

[128]Asemio. *How Tulsa Is Preserving Privacy and Sharing Data for Social Good* 2019 https://asemio.com/wp-content/uploads/2022/10/How-Tulsa-is-Preserving-Privacy-and-Sharing-Data-for-Social-Good-2.pdf Last accessed: October 2024

3 Trusted Execution Environments

Passively secure secret sharing based MPC schemes are guaranteed not to leak any information to processing parties that correctly follow the protocol. Actively secure variants of secret sharing based MPC also protect against processing parties who deviate from the protocol to attack it. In any case, confidential data are leaked if an intruder manages to break into the infrastructure of a sufficient number of processing parties. In this case they can gather all the secret shares together and reconstruct the original data.

Trusted Execution Environment (TEE) technology provides an additional layer of protection which increases the cost of successful attacks. A TEE is a virtual environment for running software with heightened security requirements, which makes intrusion even by privileged users extremely difficult. Other software on the same system cannot see what exact data is processed inside of the TEE. Remote users like data providers and other computing parties in a MPC setting can, meanwhile, cryptographically verify that only the expected software runs inside the TEE. The details of these capabilities are explained in Section 3.1.

Figure 1 illustrates the implications of such a technology for the data providers. Data providers get the assurance that their confidential data is only processed in the agreed-upon way by the pre-defined software, which prevents leaking of confidential data. In principle, the user usually "knows" which software runs on their local computer[1]. When they use classical cloud services, they have no choice but trusting that the security claims made by the cloud provider hold true, as they have no way to gain insight or detect when the "security culture was inadequate" [202].

With TEEs, one can achieve a much higher confidence in the remote software, much closer to that of local software than classic cloud services, as depicted in Figure 2.

In the context of an MPC system where computing parties run MPC software within TEE, if an attacker finds a way to break into the TEE technology *and* breaks into a sufficient number of computing parties, he might be able to extract the confidential data. To mitigate this risk, one may consider using diverse TEE technologies from different vendors across the set of computing parties. The increased level of security stems from the consideration that an exploit against a TEE from one vendor is unlikely to also work for other TEEs from other vendors. Relying on multiple TEE technologies should increase the cost of a successful attack to an impractical level, even for very sophisticated and powerful attackers.

3.1 Fundamentals of TEE Technologies

In the previous section we have claimed that TEEs are beneficial for the JOCONDE System for two distinct reasons: (1) software in TEEs can be verified, and (2) software in TEEs does not leak confidential data. In this section we present the technical foundations of these claims. We focus our attention on TEE technologies relevant for data center environments, leaving out TEEs designed for handheld devices. In our presentation we shall favour simplicity over technical accuracy, as our goal is to expose the main ideas to guide the development of the JOCONDE System in the context of the current state of the art in technology.

TEEs extensively use cryptography to protect the data and the code. The keys for the cryptographic operations themselves are managed by special hardware and software components.

[1] In practice, this might in fact only be true for tech-savvy users. Regular computer users are compelled to simply trust that the operating system and software vendors care for the privacy concerns of their users [200, 201].

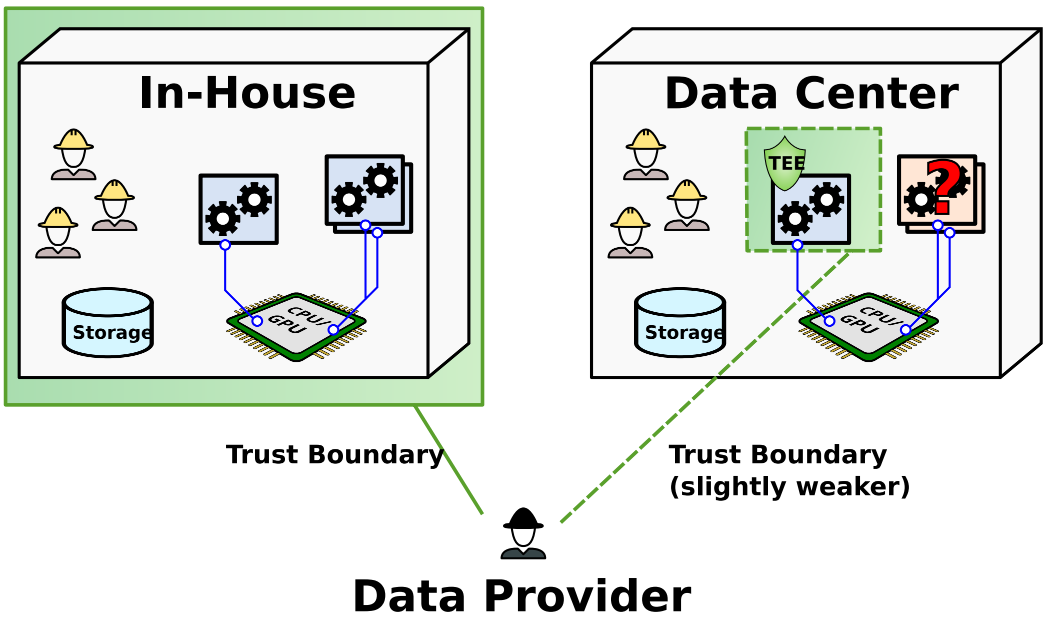


Figure 1. Data providers usually trust the software, hardware and employees within their company. TEE technology enables the data provider to extend their trust towards software running on remote servers. The data provider may have never physically inspected the physical facilities hosting the remote server. They can, however, verify that the remote machine runs the expected software within a secure TEE. TEE technology is not the ultimate solution to data protection and certain attack vectors remain possible. This implies that, in practice, trust towards an external TEE is not always as solid as the trust towards own in-house infrastructure.

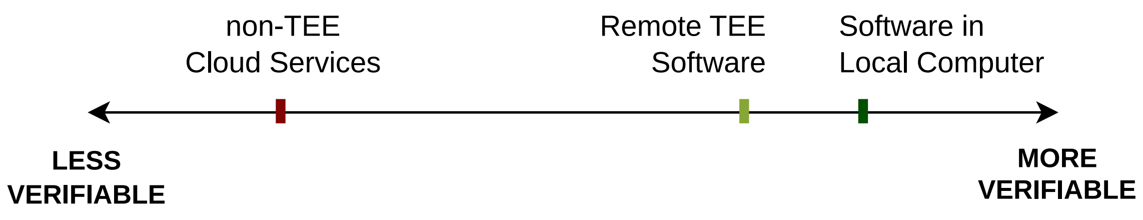


Figure 2. The user of open source software on their local computer can *in theory* verify all the locally running software and thus knows that their data is not leaked. In practice, the sheer amount of source code is impossible to inspect extensively, and the vast base of consumer grade hardware is largely not verifiable. Classical cloud services lie at the other end of this spectrum, as the only thing a user can do is trusting that the service provider lives up to their claims. Remote software *running in a TEE* can be verified thanks to cryptographic proofs during remote attestation. Some residual risks still remain, since users cannot verify the physical and virtual environment where the TEE is running in.

Other software on the same system residing outside the TEE lacks access to these encryption keys. Most of the adopted cryptographic primitives are fairly standard. The complexity lies within key management and the establishment of trust. The three core principles listed below are common across the various TEE technologies.

Protection of Working Memory (Figure 3a)
Data cannot easily leak from within a TEE. The TEE encrypts the working memory of the protected software with symmetric encryption. If anything outside of the TEE reads the working memory it will just see meaningless noise. The symmetric encryption key is only accessible by the CPU, and no mechanism exists for software to access it. When the TEE is destroyed, the encryption key is erased.

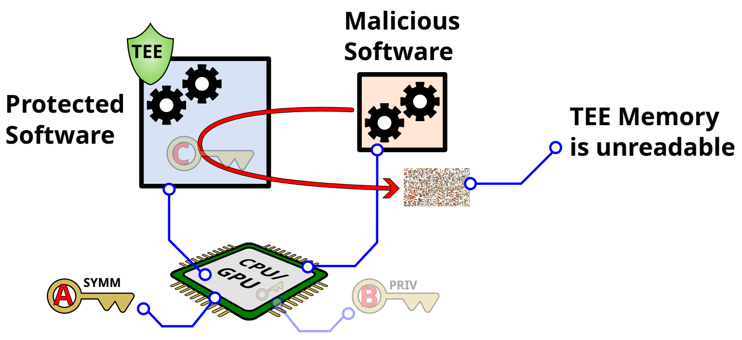
Remote Attestation (Figure 3b)
A data provider can check what software is running in the remote TEE. This process is called *remote attestation*. The underlying mechanism is simple: during the manufacturing process, the TEE vendor creates a unique secret key for each CPU which is stored in two places: (1) within the CPU, and (2) within an extra secure facility of the TEE vendor[2]. The vendor can later recognise its CPU by verifying whether the CPU contains a recorded secret key.
During remote attestation, a special component on the CPU creates a proof about the trustworthiness of the platform. The proof contains a signature over hashes, so-called *measurements*, of the protected software and some of its data. The special component creates the signature with a private key. Again, no other component on the CPU can access this private key. The TEE vendor knows the respective public key and can share it with the data provider. The data provider trusts the TEE vendor and uses the public key to verify the signature in the proof. The data provider then knows what software runs inside of the TEE and can decide whether to trust the protected software and its computing environment. Reference [203] illustrates the typical entities that are related to the concept of remote attestation and provides a common vocabulary.

Secure Data Exchange (Figure 3c)
The data provider can securely exchange data with the protected software. Secure data exchange is usually initiated with a key exchange mechanism. One challenge for the data provider is to ensure that the key exchange is indeed performed with the protected software, and not with a so called *man-in-the-middle* [204]. This is solved with remote attestation, where the signature also covers some key material which is used within the key exchange. The rest of the key exchange and secure message exchange is standard procedure, and multiple software projects indeed use the TLS cryptographic protocol (which makes HTTP*S* secure).

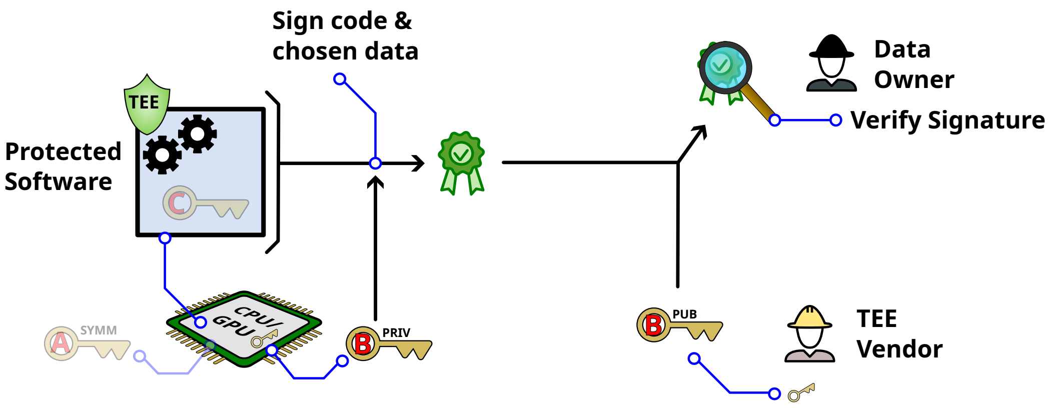
These three mechanisms allow the protection of the data during the transport phase from the data provider to the protected software, and during the processing phase in the protected software. The same measures help to ensure that only the output parties are allowed to download the computation results.

Some TEE technologies like Intel® SGX have an integrated solution for data protection across reboots of the TEE. This is called *data sealing*. However, this feature introduces new threats due to possible state rollbacks and future vulnerabilities [205, 206].

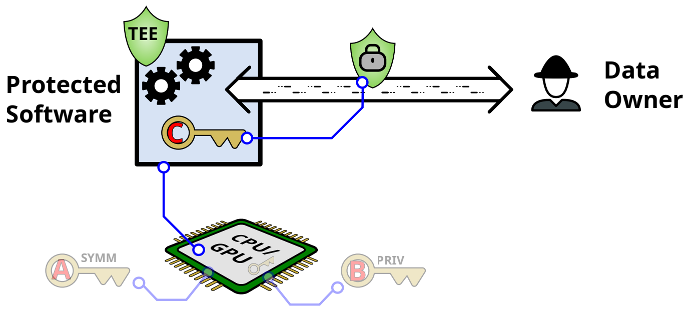
[2]A direct implication is that an end-user needs to trust the CPU vendor throughout the lifetime of the TEE solution, even after the CPU has been manufactured.



(a) One key protects the working memory of the software within the TEE from other software. This way, any extracted data just looks like meaningless noise. This key is managed by the CPU.



(b) A second key is used to establish trust in the software running inside of the TEE, and the computing environment. This key is managed by the CPU and by the TEE vendor.



(c) A third key is used to establish a secure communication channel between the data provider and the remote software in the TEE. This key is managed by the protected software.

Figure 3. Three common key types are used in TEE technology.

3.2 Security Challenges

As mentioned previously, in its current form, TEE technology provides strong security measures. Nevertheless, attackers with a sufficiently high budget might still be able to identify and exploit weaknesses. This can result in data leakage or manipulation of processing results. Figure 4 shows which attack vectors TEEs commonly attempt to protect against. The TEE is usually built upon special CPU instructions and supporting TEE components. As long as the TEE-related components and instructions themselves are correctly implemented, malicious non-TEE components cannot take control of or leak data out of the TEE. This means a malicious operating system cannot break the confidentiality of the TEE. A less capable physical attacker is also blocked off. Importantly, the protected software itself needs to be carefully written to avoid leaking secrets. Certain architecture design decisions have an influence on the attack surface of the software (Figure 5).

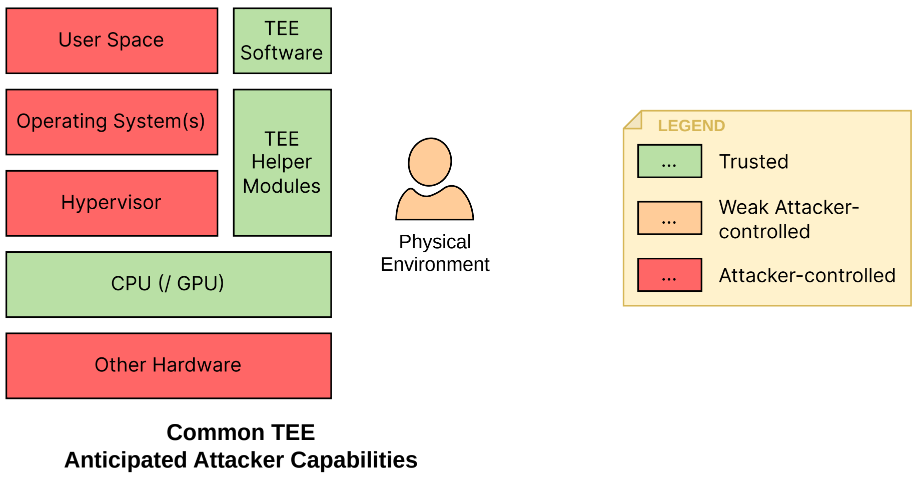


Figure 4. TEE technologies protect the enclosed software from highly privileged attackers if all TEE-related hardware, firmware, and software components work as expected. The system remains secure against an attacker who controls co-located user space software, the operating system, the hypervisor, other unrelated hardware components, and is capable of performing some basic physical attacks.

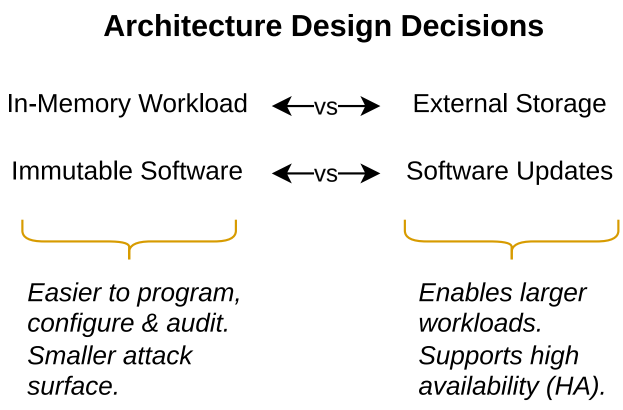

Architecture Design Decisions
In-Memory Workload
vs
External Storage
Immutable Software
vs
Software Updates
Easier to program, configure & audit. Smaller attack surface.
Enables larger workloads. Supports high availability (HA).

Figure 5. The software architecture of a service comprises multiple design decisions, like the two shown in this figure. These are trade-offs and determine in large parts the functional and quality properties of the service. The JOCONDE System may be able to use the simpler approaches for these two examples for the benefit of a smaller attack surface and easier audits.

A thorough overview of historical attacks and attack vectors is given in [207, 206, 208, 209]. The most relevant takeaways from previous literature, complemented by our direct experience, can be summarised as follows.

Vulnerable software in a TEE is still vulnerable software.
TEE technology aims to protect the enclosed software from unwanted access by the host. However, the service inside of the TEE may explicitly expose endpoints to allow the outside world to use the service. These endpoints are an attack vector and need to be carefully secured. A failure to do so puts confidential data at risk. One prominent example of such a failure is the *Log4Shell* remote-code-execution vulnerability [210], which would not have been prevented by TEE technology.

Side-channel attacks may allow attackers to sidestep the TEE protection.
Attackers use side-channel attacks to extract a very small amount of bytes out of the TEE. This attack vector remains viable even if the software inside of the TEE itself is (seemingly) free of logic errors. Side-channel attacks try to exploit *externally observable behaviour* of low-level implementation details of certain code snippets or of the hardware, to reconstruct a couple of bytes of the data which is currently being processed. The targets of such attacks are usually encryption keys, as they are small and can be used to decrypt larger volumes of confidential data.

Persisted data might be extracted by future vulnerabilities.
If today a data provider uploads confidential data, a new security vulnerability discovered in the near future could be used to exfiltrate this confidential data [206]. Hence confidential data being stored in a TEE system for a long time is under higher risk than data stored only ephemerally for a short amount of time.
Workloads in the JOCONDE System might be able to sidestep this attack vector. Its workloads are short-lived, likely lasting up to a few dozen hours. Further, they are not mission-critical, i.e. if a workload is interrupted due to a power outage, it is possible to start all over again. This means that the TEE system only lives for a short amount of time, and no cryptographic material needs to be persisted outside of the TEE to be able to recover from an interruption.

Sophisticated physical attacks are hard to shield against.
TEEs encrypt the working memory as it is stored in the RAM memory module. This protects against *cold boot* attacks where a physical attacker removes a RAM memory module and dumps its content. This is just a simple physical attack, and physical access to a CPU allows for more sophisticated attacks which are mostly out of scope of the TEE threat models.
For example, [211] shows a fault injection attack performed via physical access. A successful fault injection might alter the decision of some access control logic, or alter the output of a cryptographic algorithm to then use further cryptanalysis to access confidential data [212].
Hence it is necessary to consider the security of the physical environment, like the data center where a given CPU is located at.

How do end-users trust the software?
Data providers (and auditors) are stakeholders which need to be convinced of the system's security. This means that they need to have some trustworthy process in place for reviewing the software, deriving the expected measurement values for the remote attestation process, and securely configuring their end-user software.
Any error during the evaluation or configuration could put confidential data at risk. For example, if data providers use an external[3] website to upload their confidential data, they are incapable to reliably verify the source code of that website.

[3]"Attacker-controlled": Outside of the control of the end-user, e.g. operated by the host of the TEE service.

Software updates are complicated.

As software ages it needs to be updated. In the TEE context, this adds a risk factor if old data are to be migrated from an old version to a new version of the TEE software. Such a mechanism needs to ensure that *only an approved*, i.e. hopefully non-malicious, new version of the TEE software can access the old data. This can become technically rather involved, especially if service downtime is to be minimised. Getting the details right *and* convincing non-domain experts of the security of such an update process might be complicated. Instead, wherever possible, one should use short-lived, *ephemeral* workloads which don't require software updates.

Workloads in the JOCONDE System might be able to sidestep this attack vector. Its workloads are short-lived analyses, likely lasting up to a few dozen hours. If a security fix becomes available, the upcoming workloads can be based on the updated software components. Running workloads can be stopped or just let to run to the end, depending on the severity of the vulnerability.

We believe that finding an exploit against a TEE technology requires (1) expert knowledge with many years of experience creating low-level exploits, and (2) a good amount of sheer luck to use the right tool from the right angle in the right spot, figuratively speaking. CPUs are very complicated and various TEE-related hardware and software components are not publicly documented, meaning that attackers oftentimes need to attack a *black blox*. An exploit itself is only useful if the attacker is also able to bring the exploit to the TEE. For a physical exploit this means to break into a data center, while for a software exploit one needs to break into some virtual infrastructure [213, 214, 215]. In conclusion, *TEE technologies provide a strong layer of protection against all but the most sophisticated attackers.* We believe that the deployment process, configuration, orchestration, and exposed endpoints of the protected software itself are easier targets for an attacker.

If, however, an attacker manages to overcome all hurdles and successfully attack protected software within a TEE, he might be able to extract encryption keys of the communication channels, produce wrong results in the computation, modify some decision ("deny access" into "grant access"), or gain direct access to the raw confidential data. In the JOCONDE System, depending on how the TEE will be used, this may put the confidential data, the secret shares, or the correctness of analysis results at risk.

Security vulnerabilities in TEEs have been found both by independent security researchers and by security researchers working for TEE vendors. Security researchers follow the practice of *responsible disclosure* to give vendors enough time to fix the security vulnerabilities. Vendors have processes in place to patch the CPU and TEE components [216, 217, 218, 219]. These update processes are scrutinised as well [206, 220]. However, there have been security vulnerabilities requiring fixes at the hardware level, or – if possible at all – expensive software-level workarounds [221, 222, 223].

3.3 TEE Technologies on the Market

3.3.1 Overview

Every available TEE technology can increase the security posture of a solution, if used correctly. The TEE technologies mostly differ in architectural details and the available software ecosystem support (a software overview can be found in Section 3.4). Table 3 shows a very high-level summary of the relevant TEEs. The different TEE technologies are introduced in more detail in the following subsections. A more detailed comparison can be found in [224, 225]. Note that we only consider TEE technologies which are relevant in the data center environment. Similar isolation-focused security technologies like the very widespread Secure Enclave from Apple [226], Trust-Zone from ARM [227] etc. are considered too constrained for the JOCONDE project and hence not listed in this comparison. TEE designs for the RISC-V architecture are omitted as it is unclear when they will be widely commercially available in European cloud environments. IBM's TEE technologies Secure Execution for Linux (SE) [228, 229, 230] and Protected Execution Facility (PEF) [231] are omitted because public documentation concerning the remote attestation process is lacking, and the authors could not find a convincing method for third-party end-users to access measurements of user space applications during remote attestation.

TEE technologies can be roughly categorised into process-based, VM-based, and hybrid models.

Process-based

Process-based TEE technologies are designed to protect the part of an application (*trusted part*) which processes confidential data. In addition to this trusted part, the TEE also contains a small amount of special TEE-specific code. This means that the amount of software within the TEE, the so-called *Trusted Computing Base* (TCB), is comparatively small, and so is the attack surface.

Programming for a process-based TEE can be challenging without higher level abstractions like *Gramine* (Section 3.4). The trusted part needs to be separate from the rest of the application. Any input and output operations need to be forwarded by the *untrusted* part of the application outside of the TEE. This results in an artificially complicated architecture. Programming language selection might be restricted to languages like C and C++. Overall, developers are confronted with additional complexities for their choice of programming languages, dependencies, but also with regards to debugging and profiling capabilities. The primary process-based TEE technology as of October 2024 is Intel® SGX.

VM-based

VM-based TEE technologies protect a full Virtual Machine (VM). This means that the operating system kernel and all required user space applications and their dependencies are wholly enclosed within the TEE. A software solution relying on VM-based TEE technology has a larger TCB compared to a process-based TEE, and thus a larger attack surface. A wrong configuration or a backdoor in the kernel or a software dependency could fully circumvent the protection offered by the TEE. On the other hand, these TEE technologies add protections around a mostly standard VM. This means that developers can program for the confidential VM using a rich selection of familiar tools, languages, dependencies, processes, etc. as they would for a regular VM.

Hybrid

The AWS Nitro Enclaves represent a sort of a hybrid solution between process-based and VM-based TEEs. They provide a VM as the runtime environment but isolate the VM from the rest of the world, similar to process-based TEE technologies. The goal is to provide developers with a rich development environment while keeping the attack surface small.

The JOCONDE System will likely support multiple TEE technologies, an approach similar to [37]. As mentioned in Section 3.2, every supported TEE technology provides a significant additional security layer. As the JOCONDE System relies on distributed MPC computation, we are in the position to use multiple TEE technologies on the different computing nodes to further improve the security level of the system as a whole: an exploit identified for one specific TEE technology would likely not work for other TEE technologies due to the differences in hardware implementation and TEE design. Hence it is recommended that the JOCONDE System software is designed

Table 3. A summary of the TEE technologies considered relevant for the JOCONDE System.

TEE Technology	Verdict
Intel® TDX	*Suitability for the JOCONDE System: Suitable* Intel® TDX is the `x86-64` VM-based TEE offering from Intel®. Since an Intel® TDX VM will contain a lot of software (kernel, all of the user space applications), more work needs to be put into a security audit. However, it provides developers with a familiar runtime environment and should not add more hurdles to software development than necessary.
AMD SEV-SNP	*Suitability for the JOCONDE System: Suitable* AMD SEV-SNP is the `x86-64` VM-based TEE offering from AMD. Since an AMD SEV-SNP VM will contain a lot of software (kernel, all of the user space applications), more work needs to be put into a security audit. However, it provides developers with a familiar runtime environment and should not add more hurdles to software development than necessary.
Intel SGX	*Suitability for the JOCONDE System: Challenging* Intel® SGX is the `x86-64` Process-based TEE offering from Intel®. Its unique design shrinks the attack surface significantly, but severely restricts the developer experience. If the software for JOCONDE can be written in a way that it works on Intel® SGX, then Intel® SGX could provide a significant security improvement: it makes the TEE layer more heterogeneous and hence maybe harder to intrude compared to employing only VM-based TEE technologies.
ARM CCA	*Suitability for the JOCONDE System: Suitable when hardware becomes available* ARM CCA is the `ARM64` VM-based TEE design from ARM. No hardware with ARM CCA support exists as of October, 2024, but the authors believe that hardware will be available rather sooner than later. Since an ARM CCA VM will contain a lot of software (kernel, all of the user space applications), more work needs to be put into a security audit. However, it provides developers with a familiar runtime environment and should not add more hurdles to software development than necessary.
AWS Nitro Enclave	*Suitability for the JOCONDE System: Likely Suitable* AWS Nitro Enclave is the VM-based TEE offering from AWS, available for both `x86-64` and `ARM64`. Since an AWS Nitro Enclave VM will contain a lot of software (kernel, all of the user space applications), more work needs to be put into a security audit. However, it provides developers with a mostly familiar runtime environment, and most development friction is expected to arise due to the isolation from the network: network requests need to be forwarded by a parent EC2 instance. Similar to Intel® SGX, AWS Nitro Enclave provides a unique design and would contribute to a more heterogeneous, i.e. maybe more secure, TEE layer in the JOCONDE System.

in a way that is easily portable across various TEE technologies to reduce development costs. Table 4 presents an overview of how easily software can be ported from one TEE technology to another. We expect that targeting `x86-64` VM-based TEE technologies creates the least amount of development overhead while still achieving the intended goals of the project.

Table 4. A software solution needs to be modified when it is ported *from* one TEE technology *to* another TEE technology. This table lists the major components requiring modifications for a given port.
***ISA*: The software needs to be compiled to another Instruction Set Architecture, e.g. `x86_64` to `arm64` or vice versa; *N*: The network module; *O*: The orchestration module; *RA*: The remote attestation module; *SA*: The overall software architecture. Note: the amount of rework depends on how well the existing software solution happens to match the possibilities of the target TEE technology.**

From \ To	Intel® SGX	TDX/SEV-SNP[*]	ARM CCA	AWS Nitro Enclave[†]
Intel® SGX		N, O, RA, SA	N, O, RA, SA	N, O, RA, SA
TDX/SEV-SNP	N, O, RA, SA	RA	ISA, RA	N, O, RA
ARM CCA	ISA, N, O, RA, SA	ISA, RA		N, O, RA
AWS Nitro Enclave	ISA[‡], N, O, RA, SA	ISA[‡], N, O, RA	ISA[‡], N, O, RA	

[*]Intel® TDX and AMD SEV-SNP are similar enough that they can be merged in this table for brevity. The remote attestation module requires the most additions for porting between these TEE technologies.
[†]AWS Nitro Enclave technology is available both for `x86_64` and `arm64`; when porting to it one can therefore choose a matching architecture and does not need to compile for another ISA.
[‡]An ISA change is only necessary if the source ISA and target ISA differ.

3.3.2 Intel SGX

3.3.2.1 Overview

Intel® SGX [232, 233, 234] is a process-based TEE technology. It was first made available in 2015. Initially it targeted end-user hardware, but in recent years the focus has shifted towards enterprise and cloud environments. This section focuses only on the most recent enterprise and cloud aspects of Intel® SGX, as end-user hardware no longer includes Intel® SGX technology.

The Intel® SGX TEE is called an *enclave*. The speciality of Intel® SGX enclaves is their small TCB. Enclaves are created of just a shared object (.so) file. As a result, simpler enclaves can comprise less than $0.5\,\mathrm{MiB}$ of compiled code. This is achieved by including only the minimum amount of functionality into enclaves. When using the Intel® SGX SDK, the default way is to use the C or C++ programming languages. These languages can be embedded more easily into the enclave environment. Languages like Java and Python usually require heavier runtimes (the *Java Virtual Machine* or the *Python Interpreter*) which cannot run as-is with the Intel® SGX SDK. Additionally, an enclave cannot be run as-is but needs to be embedded in an host application. This host application needs to communicate with the storage, network peers, or perform other tasks on behalf of the enclave (Figure 6). Since the host application is outside of the enclave, the enclave code needs to assume that *the host application might be malicious*. Solutions exist which try to hide the split into host and enclave, see for example *Gramine* in Section 3.4.

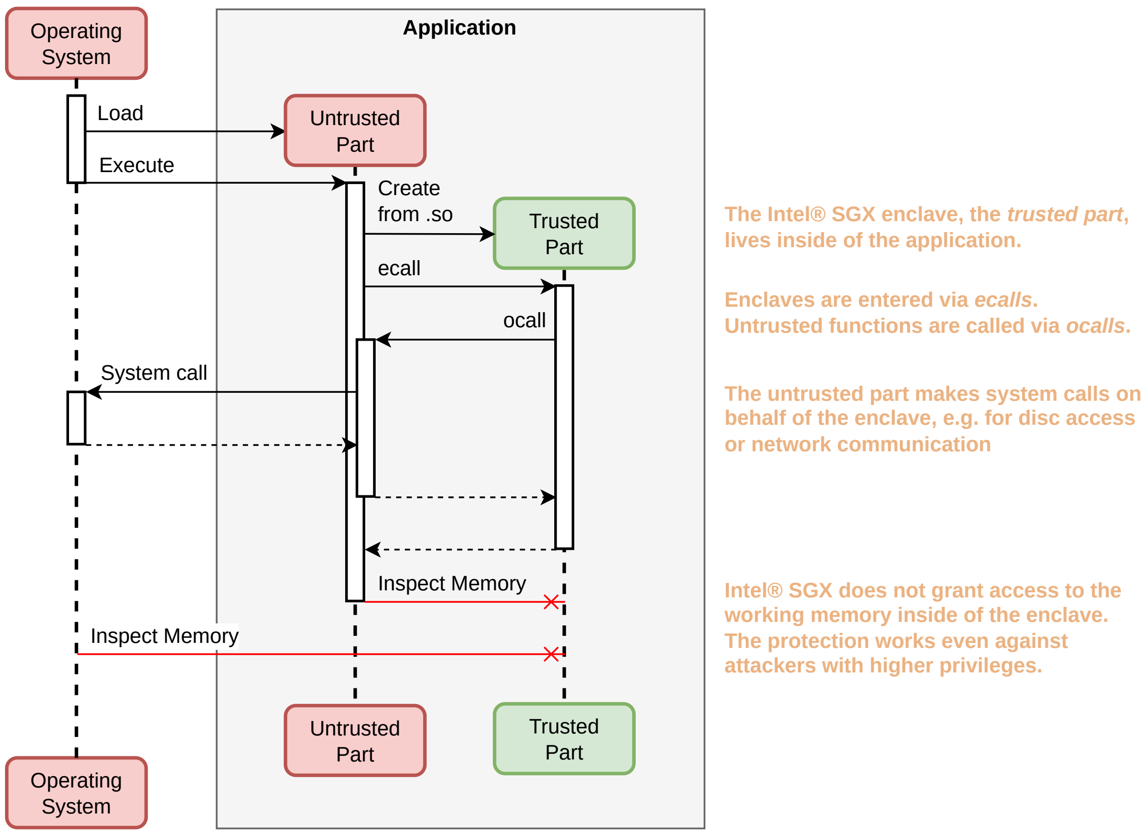

Operating System
Application
Load
Execute
Untrusted Part
Create from .so
Trusted Part
ecall
ocall
System call
Inspect Memory
Inspect Memory
Untrusted Part
Trusted Part
Operating System
The Intel® SGX enclave, the *trusted part*, lives inside of the application.
Enclaves are entered via *ecalls*. Untrusted functions are called via *ocalls*.
The untrusted part makes system calls on behalf of the enclave, e.g. for disc access or network communication
Intel® SGX does not grant access to the working memory inside of the enclave. The protection works even against attackers with higher privileges.

Figure 6. Intel® SGX enclaves have a very small TCB. This reduces the attack surface, but comes at the cost of a restricted runtime environment. Enclaves need to be embedded in a host application. (Destruction is omitted from the image for brevity.)

3.3.2.2 Properties

Auditable Codebase

The reduction of the TCB to a minimum allows the Intel® SGX specific boilerplate code (e.g. provided in the form of the Intel® SGX SDK) and the enclave part of the application to be reviewed. The code is still complicated but a lot less than in VM-based TEEs which include the Linux kernel. However, developers might choose library-OS environments like Gramine (see Section 3.4) to ease the development of enclaves, which increases the code base to review.

Runtime Identity

Due to the small size of enclaves, their identity is primarily determined by the hash of the .so file (`MRENCLAVE`). Upon enclave creation the full `.so` data is loaded into memory and hashed. It is thus trivial to detect whether two enclaves were created from the same enclave file. In contrast, most VM-based TEEs use more complicated mechanisms where this cannot be detected easily.

The Operating System Manages Threads

The untrusted operating system manages the threads which enter an enclave, and it can suspend and resume them at any time. This means that bugs in the synchronization of multi-threaded code inside of an enclave can be trivially exploited [235]. Another side effect of this were the rather powerful *single-stepping* and *zero-stepping* techniques, best explained in the paper explaining their mitigation [236].

Restricted Environment

An SGX enclave is a restricted environment. Syscalls cannot be called, and thus no direct access to e.g. logging (to console or files), thread or process management are available. This complicates development in two ways: (1) An application/the code base needs to be split into an untrusted host application and a trusted enclave. (2) One cannot use arbitrary programming languages or dependencies, as these usually expect a regular user space environment. As a countermeasure, library OS approaches like Gramine are used, however they increase the code base again and cannot solve all problems.

Data Sealing

Intel® SGX enclaves have access to an encryption key for *data sealing*. The key is derived from a unique secret fused into the CPU and of the `MRENCLAVE` value (more configuration options exist [237]). This key can be re-derived by any enclave with the same `MRENCLAVE` value even across reboots, but only on the same CPU. This feature can be used to encrypt data which can only be decrypted by the same enclave, and thus as a building block to increase availability. The data sealing capability needs to be used with care as it allows for state rollbacks and enclave "twins" [205].

3.3.3 Intel TDX

3.3.3.1 Overview

Intel® TDX [238, 239, 240, 241] is a VM-based TEE technology and the second TEE offering on Intel® server processors. It was first made available to the general public in 5th Gen Intel® Xeon® Scalable Processors in 2023. It complements Intel® SGX and uses Intel® SGX enclaves during remote attestation. Whereas Intel® TDX adds protection to unmodified VMs, the full potential can only be achieved if the guest kernel and some user space component are aware of Intel® TDX. Software developers are not restricted in their choice of technology, as long as it can run inside of a VM and complies with the threat model. From a user perspective, it is overall very similar to AMD SEV-SNP.

3.3.3.2 Properties

Large Trusted Computing Base

A VM usually contains a lot of software and may provide a large attack surface. Whereas Intel® SGX and AWS Nitro Enclave encourage keeping only security-relevant code inside the TEE, Intel® TDX allows placing all the code inside the VM. This increases the size of the Trusted Computing Base. If some arbitrary ancillary service inside of the VM is vulnerable, it could pose a threat to confidential data. Hence, considerably more code needs to be reviewed and trusted.

Runtime Identity

VMs consist of multiple pieces, and Table 5 shows different strategies for how this can be measured. Intel® TDX has multiple measurement registers for the different pieces of the VM.

Familiar Runtime Environment

Since Intel® TDX adds protections around regular VMs, developers can use their familiar software stacks for building services. As always, software dependencies need to be vetted for trustworthiness, and any software processing confidential data should be hardened against side-channel attacks.

Table 5. A traditional VM consists of multiple parts: UEFI firmware, the kernel, the kernel command line, initrd and the user space applications on some root file system. With confidential VMs, this software stack can be measured in various ways [242]. Note that confidential VMs from cloud providers do not necessarily allow all of these strategies and may provide a separate way.

User Space Measurement Strategy	Description
Service in initrd	A small and self-contained target service can be directly started as `/init` out of initrd. This is trivially covered by the measurement register(s) provided by the different TEE-VM technologies
DM-verity	The integrity of a read-only root block device can be covered by a hash tree with *DM-verity*. The root hash is provided as a kernel command line argument. Again, this is covered by the various measurement register(s).
Virtual TPM and IMA	The Linux Integrity Measurement Architecture (IMA) hashes executables and certain files upon opening and rolls up the hashes into Trusted Platform Module (TPM) registers. *Intel® TDX:* A TDX measurement register holds the rolled-up hash. *AMD SEV-SNP:* Using the new Virtual Machine Privilege Level (VMPL) feature, one can implement a virtual TPM [243].

3.3.4 AMD SEV-SNP

3.3.4.1 Overview

AMD SEV-SNP [244, 245] is a VM-based TEE technology, first available in 3rd Gen AMD EPYC™ processors since 2021. Only this iteration of the AMD SEV technology removes the hypervisor from the TCB and allows third parties to perform remote attestation. Whereas AMD SEV-SNP adds protection to unmodified VMs, the full potential can only be achieved if the guest kernel and a user space component are aware of AMD SEV-SNP. Software developers are not restricted in their choice of technology, as long as it can run inside a VM and complies with the threat model. From a user perspective, it is overall very similar to Intel® TDX.

3.3.4.2 Properties

Large TCB
A VM usually contains a lot of software and may provide a large attack surface. Whereas Intel® SGX and AWS Nitro Enclave encourage keeping only security-relevant code inside the TEE, AMD SEV-SNP allows placing all the code inside the VM. If some arbitrary ancillary service inside of the VM is vulnerable, it could pose a threat to the confidential data. Hence, a lot more code needs to be reviewed and trusted.

Runtime Identity
VMs consist of multiple pieces, and Table 5 shows different strategies for how this can be measured. AMD SEV-SNP only has a single measurement register.

Familiar Runtime Environment
Since AMD SEV-SNP adds protections around regular VMs, developers can use their familiar software stacks for building services. As always, software dependencies need to be vetted for trustworthiness, and any software processing confidential data should be hardened against side-channel attacks.

Data Sealing
AMD SEV-SNP exposes a key derivation interface similar to the one present in Intel® SGX. A sealing key can be derived from the VM launch measurement and a key unique to a given CPU (and a couple of other parameters). The same sealing key can be derived across VM reboots on the same CPU. This API is flexible and also supports other configurations for the selected key derivation material. It can be used as a building block to increase availability. The data sealing capability needs to be used with care as it allows for state rollbacks and VM "twins" [205].

3.3.5 ARM CCA

3.3.5.1 Overview

ARM CCA [246, 247] is a VM-based TEE technology, with no off-the-shelf hardware available so far. ARM added CCA in 2021 in the form of the *Realm Management Extension* to ARMv9-A. It is an additional TEE technology available on ARM chips alongside *TrustZone*. While TrustZone is geared towards OEMs to implement harden certain services of the platform, ARM CCA is similar to Intel® TDX and AMD SEV-SNP where one can move arbitrary workloads into the TEE during the runtime of the system. Both TrustZone and ARM CCA workloads are isolated from each other.

3.3.5.2 Properties

Large TCB
A VM usually contains a lot of software and may provide a large attack surface. Whereas Intel® SGX and AWS Nitro Enclave encourage keeping only security-relevant code inside the TEE, ARM CCA allows placing all the code inside the VM. If some arbitrary ancillary service inside of the VM is vulnerable, it could pose a threat to confidential data. Hence, a lot more code needs to be reviewed and trusted.

Runtime Identity
VMs consist of multiple pieces, and Table 5 shows different strategies for how this can be measured. ARM CCA has multiple measurement registers for the different pieces of the platform and VM.

Familiar Runtime Environment
Since ARM CCA adds protections around regular VMs, developers can use their familiar software stacks for building services. Software dependencies need to be vetted for trustworthiness, and any software processing confidential data should be hardened against side-channel attacks.

Implementation Defined
Specific silicon with ARM CCA support can have additional features. ARM CCA is a specification and vendors have some freedom in the implementation.

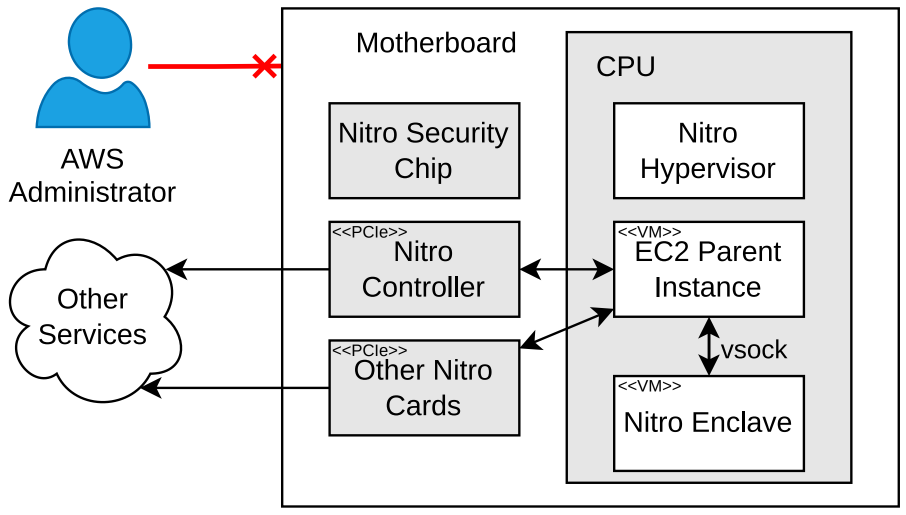


Figure 7. The AWS Nitro system consists of multiple components and can create enclaves. These are isolated VMs which can only communicate with their parent EC2 instance via a socket *vsock*. The host instance needs to manage the life-cycle and forward requests and responses to and from the enclave. The AWS Nitro system prevents administrators from manipulating system resources or looking into the running system, and isolates VMs from each other.

3.3.6 AWS Nitro Enclave

3.3.6.1 Overview

The AWS Nitro Enclave technology [218, 248, 249] is provided by the AWS Nitro system on EC2 instances within the AWS cloud, but not on-premises. The AWS Nitro system consists of specialised silicon like PCIe cards and chips (see Figure 7), which are developed internally in AWS (by Annapurna Labs, which is part of AWS). The AWS Nitro Enclave technology allows running special container images in an isolated virtual machine. The virtual machine has its own kernel and can run arbitrary applications. However, the enclave VM is isolated from the rest of the world very similar to how it is done in Intel® SGX. The enclave VM needs a parent EC2 instance which it is connected to via a socket (*vsock*). Any communication with the outside world needs to be passed to and handled by the (potentially malicious) parent EC2 instance. Conceptually, it provides a mix of Intel® SGX and Intel® TDX, using AWS' own building blocks.

3.3.6.2 Properties

Single Communication Entry Point
The AWS Nitro Hypervisor restricts the communication capabilities of the enclave severely. It can only talk to its *parent EC2 instance* via a socket *vsock*. More specifically, an enclave cannot perform arbitrary network communication or access persistent storage but needs to request assistance from its parent EC2 instance through vsock. Note that this design is similar to how Intel® SGX enclaves need assistance of the untrusted part of the application.

Improved Side-Channel Safety
The AWS Nitro Hypervisor can assign the enclave CPU cores and memory which cannot be used at the same time by other VMs on the same physical server. This categorically prevents many side-channel attacks.

Isolation from the Cloud Operator
The AWS Nitro system is designed in a way that AWS operators cannot look into an EC2 in-

stance, even if the AWS Nitro Enclave technology is not used. The software interface does not expose functionality to log into the system or extract runtime information. Encryption keys are stored in volatile hardware, inaccessible to operators. Low-level commands between hardware components are filtered by dedicated Nitro chips. Firmware updates need to be deployed through a central system with various safeguards like code reviews and signing of binaries. Firmware updates, which bypass this system, are not accepted by the AWS Nitro components.

Memory Encryption
The working memory is not encrypted by the AWS Nitro system but by the CPUs themselves. This is supported by modern server CPUs [250].

Runtime Identity
The identity of an enclave is determined by the hash of its container image, similar to Intel® SGX. The measurement values of an enclave also contain hashes covering the boot process, AWS account information and more.

Note that most of the security properties of the AWS Nitro system apply to regular EC2 instances as well. AWS Nitro Enclaves are only special in that they can be remotely attested, and that they can only communicate via vsock.

3.4 TEE Software Ecosystem

This section lists a few software projects relevant to the TEE ecosystem. The list is not exhaustive and more open-source and commercial products exist with similar capabilities, but it should give an idea of what is in general available. Since the TEE technology world is rather new, the software ecosystem is still very much in flux. Most projects support only a subset of the available TEE technologies but plan to add broader TEE support in the future.

Intel® SGX SDK
The official SDK for developing applications for Intel® SGX, developed by Intel® [251, 237]. It is written in C and C++, provides a limited C and C++ standard library for the restricted enclave environment, and a C API for accessing SGX functionalities. It is rather low level, and mostly only provides wrappers for the very low-level detail, e.g. for enclave entrance and exit, data sealing and remote attestation message processing, with a file crypto library being on the higher-level end.

Open Enclave SDK
Open Enclave SDK [252] operates on a similar level like the Intel® SGX SDK, but attempts to abstract away the low-level details of the underlying TEE technology. At the moment it supports Intel® SGX and ARM TrustZone. Its main contributor is Microsoft.

AWS Nitro Enclave SDK
The official SDK for developing applications for AWS Nitro Enclaves, developed by AWS [253]. It operates on a similar level to Intel® SGX SDK. Whereas the SDK provides a C API, any languages can be used within an AWS Nitro Enclave.

Gramine
Gramine (formerly Graphene-SGX) [254] enables common existing services to run inside a TEE. The service itself does not need to be TEE-aware and requires only minimal or no modifications at all. Gramine provides a syscall emulation layer, a so-called *library OS*. Currently it supports Intel® SGX.

Enarx

Enarx [255] provides a WASM runtime which can run inside different TEEs. A user can write their service in any language which allows compilation to WASM. The project started in 2019 and currently supports Intel® SGX, Intel® TDX and AMD SEV-SNP.

Veracruz

Veracruz [256] is a framework to analyse confidential data from multiple data providers within a TEE. Data providers attest a policy stating which workload will be run and who has what permissions. The policies are enforced within the TEE, hence data providers are assured that the TEE protects their data. Veracruz uses WebAssembly technology for the analysis code to abstract over the underlying TEE technology. It is primarily developed by Arm Research and had its last release, 22.07, in August 2022.

KubeVirt CVM

KubeVirt CVM [257] builds upon the established KubeVirt project. KubeVirt aims to integrate existing VMs without modifications into a Kubernetes cluster, and KubeVirt CVM extends it to support deployment into TEEs. The KubeVirt CVM project is rather young and is lacking support for most TEE technologies besides Intel® TDX.

Confidential Containers

If a service is orchestrated via Kubernetes, then the Confidential Containers project [258] makes it possible to schedule the services into TEEs. The Confidential Containers project is actively evolving and released version 0.9.0 in July 2024. It builds on the Kata Containers project, which is widely used in production. Note that Confidential Containers injects its own components into the TEE to enable container deployment and attestation. In contrast, KubeVirt CVM does not inject anything into the TEE.

Veraison

Veraison [259] aims to provide software components and standards for remote attestation. It is not a ready-to-run solution, as there are various ways for how remote attestation might be done in real world scenarios. Instead of inventing a remote attestation from scratch in each TEE solution, Veraison tries to provide a starting point for further modification and integration.

3.5 TEE Availability in the Cloud

TEEs are available from multiple cloud service providers (CSP). Table 6 provides an overview of which TEE technologies are available in which clouds. Note that this table is not exhaustive and just provides an initial orientation for the reader, as TEE hardware is relatively new and not yet deployed by all CSPs. CSPs usually provide TEE offerings in two distinct forms:

Bare Metal

Also called *Dedicated Server* or similar. The customer rents a physical server for their own use. The customer themselves starts the TEEs, i.e. applications using Intel® SGX or a VM with activated AMD SEV-SNP. We recommend bare metal servers for the following reasons: (1) attackers cannot rent a co-located VM, meaning that they have to break into more infrastructure to get close to the TEE for an attack, and (2) the customer can use the TEE technology to a full extent, especially regarding attestation of user space applications (see the next point).

Virtual Machine

The customer rents a VM which has been created on-demand by the CSP. These VMs may contain special components from the CSP, which will be part of the measurement but may

not be open source. This means *the customer needs to trust the CSP as well*[4] [260]. Additionally, documentation is scarce on which VM offerings support attestation up to the user space applications [260]. For example, Azure specifically mentions that user space applications on Intel® TDX cannot be attested as of October 2024 [261]. Their confidential VM offering is in a preview state (similar to GCP), so the authors expect that these shortcomings will be fixed in the near future as the VM offerings grow out of the preview state.
VMs are usually co-located with VMs from other customers on the same physical server, meaning that an attacker could get rather close to a victim TEE just by renting a VM. The protected software is then possibly more exposed to side-channel attack vectors. As a countermeasure, some CSPs allow renting a *dedicated host* where the physical server is no longer shared with other customers. Another possibility might be to rent a VM size which occupies a full server, effectively holding off other tenants from that server.
The TEE VM offerings are rather new and evolving, hence they should be reevaluated in the future when an interested party wants to rent a VM.

[4]Whether this is a problem needs to be decided by the stakeholders. These components may go through a thorough review process and thus may be protected against modification by a single intruder.

Table 6. An overview of TEE availability in various CSPs. The table is based on information publicly available on the websites of the various CSPs as of October 2024 and might not reflect the true capabilities of the CSPs, nor is every CSP listed which has a TEE offering. ARM CCA is omitted as no physical hardware is available as of October 2024.
***B*: Available as bare-metal server (a.k.a. dedicated server). *V*: Available as VM. *V+*: Available as VM on a dedicated host (a.k.a sole-tenancy), i.e. the underlying physical server is not shared with other customers of the CSP.**

	Intel® SGX	Intel® TDX	AMD SEV-SNP	AWS Nitro Enclave
Azure	B V+	V	V+	–
AWS	–	–	V	B V
IBM	B V	B V	B	–
GCP	–	V	V+	–
OVHCloud	B	B	–	–
Alibaba	V	V	–	–
OTC	B	–	–	–

Sources:
Azure: https://azure.microsoft.com/en-us/pricing/details/virtual-machines/dedicated-host/, https://azure.microsoft.com/en-us/pricing/details/virtual-machines/dedicated-host/, https://learn.microsoft.com/en-us/azure/confidential-computing/virtual-machine-solutions-sgx

AWS: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/sev-snp.html

IBM: https://cloud.ibm.com/docs/bare-metal?topic=bare-metal-bm-intel-sgx, https://cloud.ibm.com/docs/vpc?topic=vpc-about-sgx-vpc, https://cloud.ibm.com/docs/vpc?topic=vpc-profiles&interface=ui#confidential-computing-profiles, https://www.ibm.com/blog/announcement/looking-ahead-4th-gen-intel-xeon-scalable-processors-on-ibm-cloud/, https://www.ibm.com/blog/announcement/3rd-gen-amd-epyc-processors-on-ibm-cloud/

GCP: https://cloud.google.com/blog/products/identity-security/new-confidential-computing-updates-for-more-hardware-security-options, https://cloud.google.com/confidential-computing/confidential-vm/docs/supported-configurations

OVHCloud: https://us.ovhcloud.com/bare-metal/intel-software-guard-extensions/, https://corporate.ovhcloud.com/en/newsroom/news/baremetal-emerald-rapids/

Alibaba: https://www.alibabacloud.com/help/en/ecs/user-guide/build-an-sgx-encrypted-computing-environment, https://www.alibabacloud.com/help/en/ecs/user-guide/build-a-tdx-confidential-computing-environment

OTC: https://www.open-telekom-cloud.com/en/support/release-notes/bms-i7-for-confidential-computing-applications

4 Considerations for the JOCONDE System

The purpose of this document is to survey the current state of the art in secure multiparty computation and trusted execution environments that are the technological pillars of the envisioned JOCONDE System. The focus is on solutions for collaborative computations over confidential input data held by multiple parties where the computation process is distributed over multiple parties in order to avoid revealing the confidential input data to any of them. The only thing to be revealed should be the final computation result. This chapter summarises the main finding from our survey, with a focus on combining these two technologies in the JOCONDE System.

4.1 Secure Multiparty Computation

Various methods are used to achieve secure multiparty computation (MPC). Firstly, secure protocols can be based on secret sharing to protect the data confidentiality, and then interactive protocols are used to compute on these shares. Secondly, secure computation can also be done using garbled circuits where the binary circuit representing the desired computations is encrypted and evaluated obliviously gate-by-gate. All inputs to the garbled circuit are encoded to keys that are used to evaluate the encrypted circuit without revealing the values. Thirdly, homomorphic encryption, especially its variants that require multiple parties to collaborate for decryption, can be used to do all computations on encrypted data. Function secret sharing and homomorphic secret sharing are currently new components that will probably soon find more use in MPC frameworks. Note that it is common for frameworks to combine these core methods in order to achieve the most efficient protocol for a given task. For all of these core secure computation methods, specific protocols for secure computation are developed to satisfy different security properties.

This document lists various tools that implement compilers, APIs, or frameworks for secure multiparty computation. Many of the systems that are currently used for practical deployments of MPC support passive security. These could be used in combination with TEEs as discussed below. Some other tools are focused on the special case of two-party computation and, therefore, also not applicable to the JOCONDE System where the computation needs to be split across at least three parties [1]. However, various MPC frameworks are still available that could be used as a backend for the JOCONDE System. Tools like EMP, EzPC, FANNG-MPC, FRESCO, FudanMPL, MP-SPDZ, PySyft, Senate, Sharemind MPC, TF-Encrypted and XSCE are secure against active adversaries and could be used directly. Some of these, like FANNG-MPC, FudanMPL, PySyft and TF-Encrypted, are more focused on privacy-preserving machine learning but can be likely used for simple data analysis as well. In addition to the supported MPC methods, careful study of the capabilities of the platforms like support for external input and output parties should be carried out to determine if all aspects required by the JOCONDE System are supported. Carbyne Stack uses MP-SPDZ as an MPC backend and offers additional services, for example, to ensure persistent data. Similar capabilities are offered by the Sharemind MPC Application Server. In addition, Divvi Up, Virtual Data Lake, XOR and CipherCompute also support external input and output parties. The set of operations supported by these frameworks as well as their efficiency should be considered further to decide which ones would be the best fit to support the desired functionalities of the JOCONDE System. More detailed information regarding these tools can be found in Section 2.4, and an in-depth comparison of all MPC frameworks considered in this report is found in Section 2.5.

The MPC field is evolving fast and MPC tools are subject to churning: some of the tools considered in this document will likely be outdated or not supported in the near future, while others will be improved and there will be new tools emerging. Most importantly, function secret sharing for computing non-linear functions is an active but relatively new direction in MPC which is likely to find more application in future MPC frameworks, and could contribute to promote the adoption of GPU-accelerated MPC. For a data analysis platform with an MPC backend to be future-proof, it is important to be sufficiently flexible to switch the underlying MPC capabilities as needed to make use of future efficiency improvements and feature upgrades. When choosing the MPC frameworks to be supported, it is also important to consider which supplementary capabilities it has to support, for example participant authentication or persistent storage.

4.2 Trusted Execution Environments

The technical foundations for Trusted Execution Environments (TEEs) were created decades ago with Hardware Security Modules (HSMs) and Trusted Platform Modules (TPMs). From those early foundations, the technology evolved into the more easily accessible TEE technology only around 2015 with the inception of Intel® SGX and the first iteration of AMD SEV; since then, TEE technologies have only kept evolving. They provide an additional layer around workloads and allow to technically enforce rules and protect data and code from the environment. The developer experience depends on whether the specific TEE technology is process-based or VM-based. Process-based TEE technologies (Intel® SGX) protect a part of an application but restrict what can be done inside the protected part of the application, whereas VM-based TEE technologies (Intel® TDX, AMD SEV-SNP, AWS Nitro Enclave, IBM SE, IBM PEF and, once available,ARM CCA) protect a whole VM and place no restrictions on what can be run inside of the VM.

In general, all TEE technologies can contribute to a significantly increased security posture of a given solution, if used correctly. Attacks on TEEs mostly focus on side-channel attacks which try to exploit low-level, often undocumented hardware behaviour, to deduce secrets inside the TEE, e.g., keys. Other than that, the protected software itself needs to be secure, as TEE technology does not help against bugs within the protected software.

End-users can remotely attest the TEE and the protected software within the TEE to be sure that the correct program is executed. In addition to the obvious benefits of preserving data confidentiality and integrity, this gives additional assurance on the data lifecycle control. For example, the end-user can be sure that their input data is deleted[1] and is not forgotten on the remote server after use. However, the remote attestation feature also requires the user to be able to answer the questions: *What is a valid TEE? What software should run inside of the TEE?* If an end-user is tricked into trusting the malicious software, the TEE cannot protect them from data leakage. End-users need to have in-house competence to audit and compile TEE software, or have to delegate this task to trusted third-parties (e.g., contractors).

If used correctly, TEE technologies add an additional security layer. TEE technologies are usable today on almost all relevant hardware platforms, and TEE availability in the cloud is increasing.

[1] Data deletion guarantees provided by TEEs are not absolute. Confidential data written to a disk is encrypted with a hardware-protected sealing key. However, if unauthorised copy of this encrypted data is made outside of the TEE, it may become subject to future vulnerabilities, e.g., if some future attack is found that breaks the encryption implementation used by the TEE. Note however that, thanks to the MPC layer, such a future attack would be successful only if unauthorised copies are made of all data from all computing parties.

4.3 Combining Secure Multiparty Computation and Trusted Execution Environments

Combinations of MPC and TEEs are a new line of research [37]. Initially, TEEs were seen as a replacement for MPC [26, 27]. However, the setup and trust assumptions between the two technologies are sufficiently different to consider them as complementary, rather than alternative to each other. This motivates the integration of both approaches.

In principle, it is possible to use the two technologies side by side. For instance, by deploying these technologies in parallel, a TEE can be used to compute some components of MPC protocols [31, 32, 28], e.g., the precomputation phase. Hybrid protocols can be defined that choose between TEE and MPC to implement individual functions based on the specific needs of the participants [30].

Alternatively, the two technologies can be stacked on top of each other. In the MPC-over-TEE model, the role of the TEE is to enhance the security guarantees provided by the MPC protocol [36, 35]. For example, a passively secure MPC protocol ensures privacy and correctness of the computation provided that all parties follow the protocol description. For active security, the assumption that all parties follow the protocol is lifted, and it is up to the protocol to ensure that the computing parties follow the protocol, or at least that deviations from the protocol description are detected. Actively secure MPC protocols are considerably more complex than passively secure MPC ones. The integrity guarantees offered to the applications running in the TEE help to ensure that their operations conform to the protocol description. Put together, a TEE, especially through remote attestation, enables all parties to verify that the others are also following the protocol. Hence, implementing a passively secure MPC protocol through software components embedded in TEEs at computing nodes achieves security against an active adversary [35]. In other words, the integrity of the protocol is ensured by the TEE while privacy of the data is ensured by MPC.

The overlay of MPC and TEEs greatly reduces the level of trust that the users must put in the TEE since leakages from a single TEE node can not leak information to the attacker if the MPC protocol guarantees security against a passive adversary. For protocols that ensure passive security but active privacy against the protocol [33, 34], this level of security needed from TEEs can be even lower as even if an attacker somehow manages to modify the execution of TEEs the adversary can only modify the outcome of the protocol but can not learn additional information about the private inputs.

Adopting the overlay MPC-over-TEE approach, the MPC frameworks that offer passive security listed in Section 2.4 can be combined with TEEs to achieve security against active adversaries in the JOCONDE System. The list of MPC protocols that can be used in this context includes passively secure secure computation protocols from Divvi Up, EzPC, FudanMPL, Helium, JIFF, Manticore, MOTION, MP-SPDZ, PySyft, Secrecy, SecretFlow, Sequre, Sharemind MPC, TF-Encrypted, and Virtual Data Lakes. The main advantage of this combination is that passively secure MPC protocols are often more efficient than actively secure protocols, and the combination of passively secure MPC and a TEE is also likely to be very efficient. Secondly, passively secure MPC is easier to implement and is more commonly used in practice. Hence, allowing for such combinations could increase the flexibility of the JOCONDE System and allow for the integration of a more diverse set of underlying MPC frameworks.

Although a VM-based TEE has a larger attack surface compared to a process-based TEE, the former should be sufficient to provide additional security guarantees on top of MPC. It is also likely that support for VM-based TEEs, especially in the cloud, will be increasing in the near future, making them easier to use. In addition, VM-based solutions are potentially easier to merge with existing MPC frameworks as they could just run the existing implementation of the framework. Process-based TEEs can also be considered but this would likely require re-implementing some of the MPC protocols for the used TEE platform, inflating implementation and maintenance costs.

As described in Chapter 3, the strategy of diversifying TEE solutions (i.e. ensuring that each computing node adopts a different TEE technologies from the other nodes) may add an extra layer of protection. This considerably increases the attack threshold, as for a successful attack on data confidentiality, an attacker must exploit vulnerabilities across multiple independent TEE technologies and break into multiple computing parties using diverse TEEs, all at the same time.

Bibliography

[1] Fabio Ricciato. “Steps Toward a Shared Infrastructure for Multi-Party Secure Private Computing in Official Statistics”. In: *Journal of Official Statistics* 40.1 (2024), pp. 3–15. DOI: 10.1177/0282423X241235259.

[2] ISO Central Secretary. *Information security — Secure multiparty computation — Part 1: General*. en. Standard ISO/IEC DIS 4922-1. Geneva, CH: International Organization for Standardization, 2022. URL: https://www.iso.org/standard/80508.html.

[3] Arka Rai Choudhuri et al. “Fluid MPC: Secure Multiparty Computation with Dynamic Participants”. In: *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*. Ed. by Tal Malkin and Chris Peikert. Vol. 12826. Lecture Notes in Computer Science. Springer, 2021, pp. 94–123. DOI: 10.1007/978-3-030-84245-1_4. URL: https://doi.org/10.1007/978-3-030-84245-1%5C_4.

[4] Peter Bogetoft et al. “Secure Multiparty Computation Goes Live”. In: *Financial Cryptography and Data Security, 13th International Conference, FC 2009, Accra Beach, Barbados, February 23-26, 2009. Revised Selected Papers*. Ed. by Roger Dingledine and Philippe Golle. Vol. 5628. Lecture Notes in Computer Science. Springer, 2009, pp. 325–343. DOI: 10.1007/978-3-642-03549-4_20. URL: https://doi.org/10.1007/978-3-642-03549-4%5C_20.

[5] Yehuda Lindell. “Secure multiparty computation”. In: *Commun. ACM* 64.1 (2021), pp. 86–96. DOI: 10.1145/3387108. URL: https://doi.org/10.1145/3387108.

[6] Martin Hirt and Ueli M. Maurer. “Player Simulation and General Adversary Structures in Perfect Multiparty Computation”. In: *J. Cryptol.* 13.1 (2000), pp. 31–60. DOI: 10.1007/S001459910003. URL: https://doi.org/10.1007/s001459910003.

[7] Matthias Fitzi et al. “Unconditional Byzantine Agreement and Multi-party Computation Secure against Dishonest Minorities from Scratch”. In: *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*. Ed. by Lars R. Knudsen. Vol. 2332. Lecture Notes in Computer Science. Springer, 2002, pp. 482–501. DOI: 10.1007/3-540-46035-7_32. URL: https://doi.org/10.1007/3-540-46035-7%5C_32.

[8] Shafi Goldwasser and Yehuda Lindell. “Secure Multi-Party Computation without Agreement”. In: *J. Cryptol.* 18.3 (2005), pp. 247–287. DOI: 10.1007/S00145-005-0319-Z. URL: https://doi.org/10.1007/s00145-005-0319-z.

[9] Jonathan Katz and Chiu-Yuen Koo. “On expected constant-round protocols for Byzantine agreement”. In: *J. Comput. Syst. Sci.* 75.2 (2009), pp. 91–112. DOI: 10.1016/J.JCSS.2008.08.001. URL: https://doi.org/10.1016/j.jcss.2008.08.001.

[10] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. “Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract)”. In: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*. Ed. by Janos Simon. ACM, 1988, pp. 1–10. DOI: 10.1145/62212.62213. URL: https://doi.org/10.1145/62212.62213.

[11] Tal Rabin and Michael Ben-Or. "Verifiable Secret Sharing and Multiparty Protocols with Honest Majority (Extended Abstract)". In: *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*. Ed. by David S. Johnson. ACM, 1989, pp. 73–85. DOI: 10.1145/73007.73014. URL: https://doi.org/10.1145/73007.73014.

[12] Richard Cleve. "Limits on the Security of Coin Flips when Half the Processors Are Faulty (Extended Abstract)". In: *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*. Ed. by Juris Hartmanis. ACM, 1986, pp. 364–369. DOI: 10.1145/12130.12168. URL: https://doi.org/10.1145/12130.12168.

[13] Berry Schoenmakers and Meilof Veeningen. "Universally Verifiable Multiparty Computation from Threshold Homomorphic Cryptosystems". In: *Applied Cryptography and Network Security - 13th International Conference, ACNS 2015, New York, NY, USA, June 2-5, 2015, Revised Selected Papers*. Ed. by Tal Malkin et al. Vol. 9092. Lecture Notes in Computer Science. Springer, 2015, pp. 3–22. DOI: 10.1007/978-3-319-28166-7_1. URL: https://doi.org/10.1007/978-3-319-28166-7%5C_1.

[14] Carsten Baum, Ivan Damgård, and Claudio Orlandi. "Publicly Auditable Secure Multi-Party Computation". In: *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*. Ed. by Michel Abdalla and Roberto De Prisco. Vol. 8642. Lecture Notes in Computer Science. Springer, 2014, pp. 175–196. DOI: 10.1007/978-3-319-10879-7_11. URL: https://doi.org/10.1007/978-3-319-10879-7%5C_11.

[15] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. "Secure Multi-Party Computation with Identifiable Abort". In: *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*. Ed. by Juan A. Garay and Rosario Gennaro. Vol. 8617. Lecture Notes in Computer Science. Springer, 2014, pp. 369–386. DOI: 10.1007/978-3-662-44381-1_21. URL: https://doi.org/10.1007/978-3-662-44381-1%5C_21.

[16] Yuval Ishai, Rafail Ostrovsky, and Hakan Seyalioglu. "Identifying Cheaters without an Honest Majority". In: *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*. Ed. by Ronald Cramer. Vol. 7194. Lecture Notes in Computer Science. Springer, 2012, pp. 21–38. DOI: 10.1007/978-3-642-28914-9_2. URL: https://doi.org/10.1007/978-3-642-28914-9%5C_2.

[17] Robert K. Cunningham, Benjamin Fuller, and Sophia Yakoubov. "Catching MPC Cheaters: Identification and Openability". In: *Information Theoretic Security - 10th International Conference, ICITS 2017, Hong Kong, China, November 29 - December 2, 2017, Proceedings*. Ed. by Junji Shikata. Vol. 10681. Lecture Notes in Computer Science. Springer, 2017, pp. 110–134. DOI: 10.1007/978-3-319-72089-0_7. URL: https://doi.org/10.1007/978-3-319-72089-0%5C_7.

[18] O. Goldreich, S. Micali, and A. Wigderson. "How to Play ANY Mental Game". In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. STOC '87. New York, New York, USA: ACM, 1987, pp. 218–229. ISBN: 0-89791-221-7. DOI: 10.1145/28395.28420. URL: http://doi.acm.org/10.1145/28395.28420.

[19] Peter Sebastian Nordholt and Meilof Veeningen. "Minimising Communication in Honest-Majority MPC by Batchwise Multiplication Verification". In: *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*. Ed. by Bart Preneel and Frederik Vercauteren. Vol. 10892. Lecture Notes in

Computer Science. Springer, 2018, pp. 321–339. DOI: 10.1007/978-3-319-93387-0_17. URL: https://doi.org/10.1007/978-3-319-93387-0%5C_17.

[20] Jun Furukawa and Yehuda Lindell. “Two-Thirds Honest-Majority MPC for Malicious Adversaries at Almost the Cost of Semi-Honest”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. Ed. by Lorenzo Cavallaro et al. ACM, 2019, pp. 1557–1571. DOI: 10.1145/3319535.3339811. URL: https://doi.org/10.1145/3319535.3339811.

[21] Koji Chida et al. “Fast Large-Scale Honest-Majority MPC for Malicious Adversaries”. In: *J. Cryptol.* 36.3 (2023), p. 15. DOI: 10.1007/S00145-023-09453-7. URL: https://doi.org/10.1007/s00145-023-09453-7.

[22] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. “Founding Cryptography on Oblivious Transfer - Efficiently”. In: *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*. Ed. by David A. Wagner. Vol. 5157. Lecture Notes in Computer Science. Springer, 2008, pp. 572–591. DOI: 10.1007/978-3-540-85174-5_32. URL: https://doi.org/10.1007/978-3-540-85174-5%5C_32.

[23] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. “Fair and Robust Multi-party Computation Using a Global Transaction Ledger”. In: *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. Lecture Notes in Computer Science. Springer, 2016, pp. 705–734. DOI: 10.1007/978-3-662-49896-5_25. URL: https://doi.org/10.1007/978-3-662-49896-5%5C_25.

[24] Joseph I. Choi and Kevin R. B. Butler. “Secure Multiparty Computation and Trusted Hardware: Examining Adoption Challenges and Opportunities”. In: *Secur. Commun. Networks* 2019 (2019), 1368905:1–1368905:28. DOI: 10.1155/2019/1368905. URL: https://doi.org/10.1155/2019/1368905.

[25] Xiaoguo Li et al. “A Survey of Secure Computation Using Trusted Execution Environments”. In: *CoRR* abs/2302.12150 (2023). DOI: 10.48550/ARXIV.2302.12150. arXiv: 2302.12150. URL: https://doi.org/10.48550/arXiv.2302.12150.

[26] Raad Bahmani et al. “Secure Multiparty Computation from SGX”. In: *Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers*. Ed. by Aggelos Kiayias. Vol. 10322. Lecture Notes in Computer Science. Springer, 2017, pp. 477–497. DOI: 10.1007/978-3-319-70972-7_27. URL: https://doi.org/10.1007/978-3-319-70972-7%5C_27.

[27] Susanne Felsen et al. “Secure and Private Function Evaluation with Intel SGX”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop, CCSW@CCS 2019, London, UK, November 11, 2019*. Ed. by Radu Sion and Charalampos Papamanthou. ACM, 2019, pp. 165–181. DOI: 10.1145/3338466.3358919. URL: https://doi.org/10.1145/3338466.3358919.

[28] Philipp Muth and Stefan Katzenbeisser. *Assisted MPC*. Cryptology ePrint Archive, Paper 2022/1453. 2022. URL: https://eprint.iacr.org/2022/1453.

[29] Pengzhi Huang et al. “STAMP: Lightweight TEE-Assisted MPC for Efficient Privacy-Preserving Machine Learning”. In: *CoRR* abs/2210.10133 (2022). DOI: 10.48550/ARXIV.2210.10133. arXiv: 2210.10133. URL: https://doi.org/10.48550/arXiv.2210.10133.

[30] Pengfei Wu et al. “Hybrid Trust Multi-party Computation with Trusted Execution Environment”. In: *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022*. The Internet Society, 2022. URL: https://www.ndss-symposium.org/ndss-paper/auto-draft-222/.

[31] Joseph I. Choi et al. “A Hybrid Approach to Secure Function Evaluation using SGX”. In: *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2019, Auckland, New Zealand, July 09-12, 2019*. Ed. by Steven D. Galbraith et al. ACM, 2019, pp. 100–113. DOI: 10.1145/3321705.3329835. URL: https://doi.org/10.1145/3321705.3329835.

[32] Brandon Broadnax et al. “Fortified Multi-Party Computation: Taking Advantage of Simple Secure Hardware Modules”. In: *Proc. Priv. Enhancing Technol.* 2021.4 (2021), pp. 312–338. DOI: 10.2478/POPETS-2021-0072. URL: https://doi.org/10.2478/popets-2021-0072.

[33] Daniel Genkin et al. “Circuits resilient to additive attacks with applications to secure computation”. In: *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*. Ed. by David B. Shmoys. ACM, 2014, pp. 495–504. DOI: 10.1145/2591796.2591861. URL: https://doi.org/10.1145/2591796.2591861.

[34] Martin Pettai and Peeter Laud. “Automatic Proofs of Privacy of Secure Multi-party Computation Protocols against Active Adversaries”. In: *IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015*. IEEE, 2015, pp. 75–89. DOI: 10.1109/CSF.2015.13. URL: http://dx.doi.org/10.1109/CSF.2015.13.

[35] Nishant Kumar et al. “CrypTFlow: Secure TensorFlow Inference”. In: *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*. IEEE, 2020, pp. 336–353. DOI: 10.1109/SP40000.2020.00092. URL: https://doi.org/10.1109/SP40000.2020.00092.

[36] Arka Rai Choudhuri et al. “Fairness in an Unfair World: Fair Multiparty Computation from Public Bulletin Boards”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. Ed. by Bhavani Thuraisingham et al. ACM, 2017, pp. 719–728. DOI: 10.1145/3133956.3134092. URL: https://doi.org/10.1145/3133956.3134092.

[37] Yehuda Lindell et al. *The Deployment Dilemma: Merits & Challenges of Deploying MPC*. https://mpc.cs.berkeley.edu/blog/deployment-dilemma.html. [Accessed 01-10-2024]. Sept. 2023.

[38] Donald Beaver. “Efficient Multiparty Protocols Using Circuit Randomization”. In: *Advances in Cryptology - CRYPTO ’91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*. Ed. by Joan Feigenbaum. Vol. 576. Lecture Notes in Computer Science. Springer, 1991, pp. 420–432. DOI: 10.1007/3-540-46766-1_34. URL: https://doi.org/10.1007/3-540-46766-1%5C_34.

[39] G. R. Blakley. “Safeguarding cryptographic keys”. In: *1979 International Workshop on Managing Requirements Knowledge, MARK 1979, New York, NY, USA, June 4-7, 1979*. IEEE, 1979, pp. 313–318. DOI: 10.1109/MARK.1979.8817296. URL: https://doi.org/10.1109/MARK.1979.8817296.

[40] Adi Shamir. “How to Share a Secret”. In: *Commun. ACM* 22.11 (Nov. 1979), pp. 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176. URL: http://doi.acm.org/10.1145/359168.359176.

[41] Daniel Escudero. *An Introduction to Secret-Sharing-Based Secure Multiparty Computation*. Cryptology ePrint Archive, Report 2022/062. 2022. URL: %5Curl%7Bhttps://ia.cr/2022/062%7D.

[42] Elette Boyle, Niv Gilboa, and Yuval Ishai. "Function Secret Sharing". English. In: *Advances in Cryptology - EUROCRYPT 2015*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2015, pp. 337–367. ISBN: 978-3-662-46802-9. DOI: 10.1007/978-3-662-46803-6_12. URL: http://dx.doi.org/10.1007/978-3-662-46803-6_12.

[43] Elette Boyle et al. "Information-Theoretic Distributed Point Functions". In: *3rd Conference on Information-Theoretic Cryptography, ITC 2022, July 5-7, 2022, Cambridge, MA, USA*. Ed. by Dana Dachman-Soled. Vol. 230. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 17:1–17:14. DOI: 10.4230/LIPICS.ITC.2022.17. URL: https://doi.org/10.4230/LIPIcs.ITC.2022.17.

[44] Junru Li, Pengzhen Ke, and Liang Feng Zhang. "Efficient Information-Theoretic Distributed Point Function with General Output Groups". In: *IACR Cryptol. ePrint Arch.* (2023), p. 625. URL: https://eprint.iacr.org/2023/625.

[45] Stanislav Kruglik et al. "Verifiable Information-Theoretic Function Secret Sharing". In: *IACR Cryptol. ePrint Arch.* (2024), p. 453. URL: https://eprint.iacr.org/2024/453.

[46] Elette Boyle et al. "Foundations of Homomorphic Secret Sharing". In: *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*. Ed. by Anna R. Karlin. Vol. 94. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 21:1–21:21. DOI: 10.4230/LIPICS.ITCS.2018.21. URL: https://doi.org/10.4230/LIPIcs.ITCS.2018.21.

[47] Quang Dao et al. "Multi-party Homomorphic Secret Sharing and Sublinear MPC from Sparse LPN". In: *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part II*. Ed. by Helena Handschuh and Anna Lysyanskaya. Vol. 14082. Lecture Notes in Computer Science. Springer, 2023, pp. 315–348. DOI: 10.1007/978-3-031-38545-2_11. URL: https://doi.org/10.1007/978-3-031-38545-2%5C_11.

[48] Andrew C. Yao. "Protocols for Secure Computations". In: *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*. SFCS '82. Washington, DC, USA: IEEE Computer Society, 1982, pp. 160–164. DOI: 10.1109/SFCS.1982.88. URL: http://dx.doi.org/10.1109/SFCS.1982.88.

[49] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. "Foundations of Garbled Circuits". In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. CCS '12. Raleigh, North Carolina, USA: ACM, 2012, pp. 784–796. ISBN: 978-1-4503-1651-4. DOI: 10.1145/2382196.2382279. URL: http://doi.acm.org/10.1145/2382196.2382279.

[50] Andreas Holzer et al. "Secure two-party computations in ANSI C". In: *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*. Ed. by Ting Yu, George Danezis, and Virgil D. Gligor. ACM, 2012, pp. 772–783. DOI: 10.1145/2382196.2382278. URL: https://doi.org/10.1145/2382196.2382278.

[51] Siam U. Hussain et al. "TinyGarble2: Smart, Efficient, and Scalable Yao's Garble Circuit". In: *PPMLP'20: Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice, Virtual Event, USA, November, 2020*. Ed. by Benyu Zhang et al. ACM, 2020, pp. 65–67. DOI: 10.1145/3411501.3419433. URL: https://doi.org/10.1145/3411501.3419433.

[52] Vladimir Kolesnikov and Thomas Schneider. “Improved Garbled Circuit: Free XOR Gates and Applications”. In: *Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Part II*. ICALP ’08. Reykjavik, Iceland: Springer-Verlag, 2008, pp. 486–498. ISBN: 978-3-540-70582-6. DOI: 10.1007/978-3-540-70583-3_40. URL: http://dx.doi.org/10.1007/978-3-540-70583-3_40.

[53] Jesper Buus Nielsen and Claudio Orlandi. “LEGO for two-party secure computation”. In: *In Theory of Cryptography (TCC’09), volume 5444 of LNCS*. Springer, 2009, pp. 368–386.

[54] Yehuda Lindell. “Fast Cut-and-Choose Based Protocols for Malicious and Covert Adversaries”. In: *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference*. Vol. 8043. Lecture Notes in Computer Science. Springer, 2013, pp. 1–17. DOI: 10.1007/978-3-642-40084-1_1. URL: http://dx.doi.org/10.1007/978-3-642-40084-1_1.

[55] Pascal Paillier. “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes”. In: *Advances in Cryptology - EUROCRYPT ’99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*. Ed. by Jacques Stern. Vol. 1592. Lecture Notes in Computer Science. Springer, 1999, pp. 223–238. DOI: 10.1007/3-540-48910-X_16. URL: https://doi.org/10.1007/3-540-48910-X%5C_16.

[56] Craig Gentry. “Fully homomorphic encryption using ideal lattices”. In: *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*. Ed. by Michael Mitzenmacher. ACM, 2009, pp. 169–178. DOI: 10.1145/1536414.1536440. URL: https://doi.org/10.1145/1536414.1536440.

[57] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. “(Leveled) fully homomorphic encryption without bootstrapping”. In: *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*. Ed. by Shafi Goldwasser. ACM, 2012, pp. 309–325. DOI: 10.1145/2090236.2090262. URL: https://doi.org/10.1145/2090236.2090262.

[58] Jung Hee Cheon et al. “Homomorphic Encryption for Arithmetic of Approximate Numbers”. In: *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10624. Lecture Notes in Computer Science. Springer, 2017, pp. 409–437. DOI: 10.1007/978-3-319-70694-8_15. URL: https://doi.org/10.1007/978-3-319-70694-8%5C_15.

[59] Zvika Brakerski. “Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP”. In: *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Lecture Notes in Computer Science. Springer, 2012, pp. 868–886. DOI: 10.1007/978-3-642-32009-5_50. URL: https://doi.org/10.1007/978-3-642-32009-5%5C_50.

[60] Junfeng Fan and Frederik Vercauteren. “Somewhat Practical Fully Homomorphic Encryption”. In: *IACR Cryptol. ePrint Arch.* (2012), p. 144. URL: http://eprint.iacr.org/2012/144.

[61] Léo Ducas and Daniele Micciancio. “FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second”. In: *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. Lecture Notes in Computer Science. Springer, 2015, pp. 617–640. DOI: 10.1007/978-3-662-46800-5_24. URL: https://doi.org/10.1007/978-3-662-46800-5%5C_24.

[62] Ilaria Chillotti et al. “TFHE: Fast Fully Homomorphic Encryption Over the Torus”. In: *J. Cryptol.* 33.1 (2020), pp. 34–91. DOI: 10.1007/S00145-019-09319-X. URL: https://doi.org/10.1007/s00145-019-09319-x.

[63] Jung Hee Cheon et al. “Homomorphic Encryption for Arithmetic of Approximate Numbers”. In: *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10624. Lecture Notes in Computer Science. Springer, 2017, pp. 409–437. DOI: 10.1007/978-3-319-70694-8_15. URL: https://doi.org/10.1007/978-3-319-70694-8%5C_15.

[64] Yongwoo Lee et al. “Efficient FHEW Bootstrapping with Small Evaluation Keys, and Applications to Threshold Homomorphic Encryption”. In: *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part III*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14006. Lecture Notes in Computer Science. Springer, 2023, pp. 227–256. DOI: 10.1007/978-3-031-30620-4_8. URL: https://doi.org/10.1007/978-3-031-30620-4%5C_8.

[65] Rikke Bendlin et al. “Semi-homomorphic Encryption and Multiparty Computation”. In: *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*. Ed. by Kenneth G. Paterson. Vol. 6632. Lecture Notes in Computer Science. Springer, 2011, pp. 169–188. DOI: 10.1007/978-3-642-20465-4_11. URL: https://doi.org/10.1007/978-3-642-20465-4%5C_11.

[66] Luis Bernardo Pulido-Gaytan et al. “A Survey on Privacy-Preserving Machine Learning with Fully Homomorphic Encryption”. In: *High Performance Computing - 7th Latin American Conference, CARLA 2020, Cuenca, Ecuador, September 2-4, 2020, Revised Selected Papers*. Ed. by Sergio Nesmachnow, Harold Castro, and Andrei Tchernykh. Vol. 1327. Communications in Computer and Information Science. Springer, 2020, pp. 115–129. DOI: 10.1007/978-3-030-68035-0_9. URL: https://doi.org/10.1007/978-3-030-68035-0%5C_9.

[67] Suhel Sayyad et al. “An Exhaustive Survey on Privacy Preserving Machine Learning using Homomorphic Encryption and Secure Multiparty Computation Techniques”. In: *Journal of Computational Analysis and Applications (JoCAAA)* 33.05 (2024), pp. 636–648.

[68] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. “On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption”. In: *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*. Ed. by Howard J. Karloff and Toniann Pitassi. ACM, 2012, pp. 1219–1234. DOI: 10.1145/2213977.2214086. URL: https://doi.org/10.1145/2213977.2214086.

[69] Alessandro N. Baccarini, Marina Blanton, and Chen Yuan. “Multi-Party Replicated Secret Sharing over a Ring with Applications to Privacy-Preserving Machine Learning”. In: *Proc. Priv. Enhancing Technol.* 2023.1 (2023), pp. 608–626. DOI: 10.56553/POPETS-2023-0035. URL: https://doi.org/10.56553/popets-2023-0035.

[70] Toshinori Araki et al. “High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by Edgar R. Weippl et al. ACM, 2016, pp. 805–817. DOI: 10.1145/2976749.2978331. URL: https://doi.org/10.1145/2976749.2978331.

[71] Ivan Damgård et al. "Multiparty Computation from Somewhat Homomorphic Encryption". In: *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference*. 2012, pp. 643–662. DOI: 10.1007/978-3-642-32009-5_38. URL: http://dx.doi.org/10.1007/978-3-642-32009-5_38.

[72] Ivan Damgård et al. "Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits". In: *Computer Security - ESORICS 2013 - 18th European Symposium on Research*. Vol. 8134. Lecture Notes in Computer Science. Springer, 2013, pp. 1–18. DOI: 10.1007/978-3-642-40203-6_1. URL: http://dx.doi.org/10.1007/978-3-642-40203-6_1.

[73] Emmanuela Orsini. "Efficient, Actively Secure MPC with a Dishonest Majority: A Survey". In: *Arithmetic of Finite Fields - 8th International Workshop, WAIFI 2020, Rennes, France, July 6-8, 2020, Revised Selected and Invited Papers*. Ed. by Jean-Claude Bajard and Alev Topuzoglu. Vol. 12542. Lecture Notes in Computer Science. Springer, 2020, pp. 42–71. DOI: 10.1007/978-3-030-68869-1_3. URL: https://doi.org/10.1007/978-3-030-68869-1%5C_3.

[74] Dragos Rotaru et al. "Actively Secure Setup for SPDZ". In: *J. Cryptol.* 35.1 (2022), p. 5. DOI: 10.1007/S00145-021-09416-W. URL: https://doi.org/10.1007/s00145-021-09416-w.

[75] Damiano Abram and Peter Scholl. "Low-Communication Multiparty Triple Generation for SPDZ from Ring-LPN". In: *Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8-11, 2022, Proceedings, Part I*. Ed. by Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe. Vol. 13177. Lecture Notes in Computer Science. Springer, 2022, pp. 221–251. DOI: 10.1007/978-3-030-97121-2_9. URL: https://doi.org/10.1007/978-3-030-97121-2%5C_9.

[76] Bart Veldhuizen et al. "Extending the Security of SPDZ with Fairness". In: *Proc. Priv. Enhancing Technol.* 2024.2 (2024), pp. 330–350. DOI: 10.56553/POPETS-2024-0053. URL: https://doi.org/10.56553/popets-2024-0053.

[77] Aner Ben-Efraim, Michael Nielsen, and Eran Omri. "Turbospeedz: Double Your Online SPDZ! Improving SPDZ Using Function Dependent Preprocessing". In: *Applied Cryptography and Network Security - 17th International Conference, ACNS 2019, Bogota, Colombia, June 5-7, 2019, Proceedings*. Ed. by Robert H. Deng et al. Vol. 11464. Lecture Notes in Computer Science. Springer, 2019, pp. 530–549. DOI: 10.1007/978-3-030-21568-2_26. URL: https://doi.org/10.1007/978-3-030-21568-2%5C_26.

[78] Ronald Cramer et al. "SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for Dishonest Majority". In: *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10992. Lecture Notes in Computer Science. Springer, 2018, pp. 769–798. DOI: 10.1007/978-3-319-96881-0_26. URL: https://doi.org/10.1007/978-3-319-96881-0%5C_26.

[79] Ivan Damgård et al. "New Primitives for Actively-Secure MPC over Rings with Applications to Private Machine Learning". In: *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 1102–1120. DOI: 10.1109/SP.2019.00078. URL: https://doi.org/10.1109/SP.2019.00078.

[80] Ivan Damgård et al. “The TinyTable Protocol for 2-Party Secure Computation, or: Gate-Scrambling Revisited”. In: *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10401. Lecture Notes in Computer Science. Springer, 2017, pp. 167–187. DOI: 10.1007/978-3-319-63688-7_6. URL: https://doi.org/10.1007/978-3-319-63688-7%5C_6.

[81] Jesper Buus Nielsen et al. “A New Approach to Practical Active-Secure Two-Party Computation”. In: *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference*. 2012, pp. 681–700. DOI: 10.1007/978-3-642-32009-5_40. URL: http://dx.doi.org/10.1007/978-3-642-32009-5_40.

[82] Enrique Larraia, Emmanuela Orsini, and Nigel P. Smart. “Dishonest Majority Multi-Party Computation for Binary Circuits”. In: *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference*. 2014, pp. 495–512. DOI: 10.1007/978-3-662-44381-1_28. URL: http://dx.doi.org/10.1007/978-3-662-44381-1_28.

[83] Sai Sheshank Burra et al. “High-Performance Multi-party Computation for Binary Circuits Based on Oblivious Transfer”. In: *J. Cryptol.* 34.3 (2021), p. 34. DOI: 10.1007/S00145-021-09403-1. URL: https://doi.org/10.1007/s00145-021-09403-1.

[84] Gilad Asharov and Yehuda Lindell. “A Full Proof of the BGW Protocol for Perfectly Secure Multiparty Computation”. In: *J. Cryptol.* 30.1 (2017), pp. 58–151. DOI: 10.1007/S00145-015-9214-4. URL: https://doi.org/10.1007/s00145-015-9214-4.

[85] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. “Completeness theorems for non-cryptographic fault-tolerant distributed computation”. In: *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. Ed. by Oded Goldreich. ACM, 2019, pp. 351–371. DOI: 10.1145/3335741.3335756. URL: https://doi.org/10.1145/3335741.3335756.

[86] Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. “General Secure Multi-party Computation from any Linear Secret-Sharing Scheme”. In: *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*. Ed. by Bart Preneel. Vol. 1807. Lecture Notes in Computer Science. Springer, 2000, pp. 316–334. DOI: 10.1007/3-540-45539-6_22. URL: https://doi.org/10.1007/3-540-45539-6%5C_22.

[87] Elette Boyle, Niv Gilboa, and Yuval Ishai. “Secure Computation with Preprocessing via Function Secret Sharing”. In: *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part I*. Ed. by Dennis Hofheinz and Alon Rosen. Vol. 11891. Lecture Notes in Computer Science. Springer, 2019, pp. 341–371. DOI: 10.1007/978-3-030-36030-6_14. URL: https://doi.org/10.1007/978-3-030-36030-6%5C_14.

[88] Elette Boyle et al. “Function Secret Sharing for Mixed-Mode and Fixed-Point Secure Computation”. In: *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part II*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12697. Lecture Notes in Computer Science. Springer, 2021, pp. 871–900. DOI: 10.1007/978-3-030-77886-6_30. URL: https://doi.org/10.1007/978-3-030-77886-6%5C_30.

[89] Sameer Wagh. “Pika: Secure Computation using Function Secret Sharing over Rings”. In: *Proc. Priv. Enhancing Technol.* 2022.4 (2022), pp. 351–377. DOI: 10.56553/POPETS-2022-0113. URL: https://doi.org/10.56553/popets-2022-0113.

[90] Benny Pinkas et al. “Secure Two-Party Computation Is Practical”. In: *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*. Ed. by Mitsuru Matsui. Vol. 5912. Lecture Notes in Computer Science. Springer, 2009, pp. 250–267. DOI: 10.1007/978-3-642-10366-7_15. URL: https://doi.org/10.1007/978-3-642-10366-7%5C_15.

[91] Samee Zahur, Mike Rosulek, and David Evans. “Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits Using Half Gates”. In: *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. Lecture Notes in Computer Science. Springer, 2015, pp. 220–250. DOI: 10.1007/978-3-662-46803-6_8. URL: https://doi.org/10.1007/978-3-662-46803-6%5C_8.

[92] Mike Rosulek and Lawrence Roy. “Three Halves Make a Whole? Beating the Half-Gates Lower Bound for Garbled Circuits”. In: *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part I*. Ed. by Tal Malkin and Chris Peikert. Vol. 12825. Lecture Notes in Computer Science. Springer, 2021, pp. 94–124. DOI: 10.1007/978-3-030-84242-0_5. URL: https://doi.org/10.1007/978-3-030-84242-0%5C_5.

[93] Chunghun Baek and Taechan Kim. *Can We Beat Three Halves Lower Bound?: (Im)Possibility of Reducing Communication Cost for Garbled Circuits*. Cryptology ePrint Archive, Paper 2024/803. 2024. URL: https://eprint.iacr.org/2024/803.

[94] David Heath and Vladimir Kolesnikov. “Stacked Garbling - Garbled Circuit Proportional to Longest Execution Path”. In: *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12171. Lecture Notes in Computer Science. Springer, 2020, pp. 763–792. DOI: 10.1007/978-3-030-56880-1_27. URL: https://doi.org/10.1007/978-3-030-56880-1%5C_27.

[95] David Heath and Vladimir Kolesnikov. “sf LogStack: Stacked Garbling with O(b log b) Computation”. In: *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part III*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12698. Lecture Notes in Computer Science. Springer, 2021, pp. 3–32. DOI: 10.1007/978-3-030-77883-5_1. URL: https://doi.org/10.1007/978-3-030-77883-5%5C_1.

[96] Donald Beaver, Silvio Micali, and Phillip Rogaway. “The Round Complexity of Secure Protocols (Extended Abstract)”. In: *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*. Ed. by Harriet Ortiz. ACM, 1990, pp. 503–513. DOI: 10.1145/100216.100287. URL: https://doi.org/10.1145/100216.100287.

[97] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. “Optimizing Semi-Honest Secure Multiparty Computation for the Internet”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by Edgar R. Weippl et al. ACM, 2016, pp. 578–590. DOI: 10.1145/2976749.2978347. URL: https://doi.org/10.1145/2976749.2978347.

[98] Yehuda Lindell and Benny Pinkas. “An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries”. In: *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*. Ed. by Moni Naor. Vol. 4515. Lecture Notes in Computer Science. Springer, 2007, pp. 52–78. DOI: 10.1007/978-3-540-72540-4_4. URL: https://doi.org/10.1007/978-3-540-72540-4%5C_4.

[99] Tore Kasper Frederiksen et al. “MiniLEGO: Efficient Secure Two-Party Computation from General Assumptions”. In: *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Vol. 7881. Lecture Notes in Computer Science. Springer, 2013, pp. 537–556. DOI: 10.1007/978-3-642-38348-9_32. URL: http://dx.doi.org/10.1007/978-3-642-38348-9_32.

[100] Tore Kasper Frederiksen et al. *TinyLEGO: An Interactive Garbling Scheme for Maliciously Secure Two-party Computation*. Cryptology ePrint Archive, Report 2015/309. 2015.

[101] Vladimir Kolesnikov et al. “DUPLO: Unifying Cut-and-Choose for Garbled Circuits”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. Ed. by Bhavani Thuraisingham et al. ACM, 2017, pp. 3–20. DOI: 10.1145/3133956.3133991. URL: https://doi.org/10.1145/3133956.3133991.

[102] Gabrielle Beck et al. “Scalable Multiparty Garbling”. In: *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*. Ed. by Weizhi Meng et al. ACM, 2023, pp. 2158–2172. DOI: 10.1145/3576915.3623132. URL: https://doi.org/10.1145/3576915.3623132.

[103] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. “Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. Ed. by Bhavani Thuraisingham et al. ACM, 2017, pp. 21–37. DOI: 10.1145/3133956.3134053. URL: https://doi.org/10.1145/3133956.3134053.

[104] Jonathan Katz et al. “Optimizing Authenticated Garbling for Faster Secure Two-Party Computation”. In: *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10993. Lecture Notes in Computer Science. Springer, 2018, pp. 365–391. DOI: 10.1007/978-3-319-96878-0_13. URL: https://doi.org/10.1007/978-3-319-96878-0%5C_13.

[105] Samuel Dittmer et al. “Authenticated Garbling from Simple Correlations”. In: *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13510. Lecture Notes in Computer Science. Springer, 2022, pp. 57–87. DOI: 10.1007/978-3-031-15985-5_3. URL: https://doi.org/10.1007/978-3-031-15985-5%5C_3.

[106] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. “Global-Scale Secure Multiparty Computation”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. Ed. by Bhavani Thuraisingham et al. ACM, 2017, pp. 39–56. DOI: 10.1145/3133956.3133979. URL: https://doi.org/10.1145/3133956.3133979.

[107] Gilad Asharov et al. “Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE”. In: *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. Lecture Notes in Computer Science. Springer, 2012, pp. 483–501. DOI: 10.1007/978-3-642-29011-4_29. URL: https://doi.org/10.1007/978-3-642-29011-4%5C_29.

[108] Pratyay Mukherjee and Daniel Wichs. “Two Round Multiparty Computation via Multi-key FHE”. In: *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. Lecture Notes in Computer Science. Springer, 2016, pp. 735–763. DOI: 10.1007/978-3-662-49896-5_26. URL: https://doi.org/10.1007/978-3-662-49896-5%5C_26.

[109] Zvika Brakerski, Shai Halevi, and Antigoni Polychroniadou. “Four Round Secure Computation Without Setup”. In: *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*. Ed. by Yael Kalai and Leonid Reyzin. Vol. 10677. Lecture Notes in Computer Science. Springer, 2017, pp. 645–677. DOI: 10.1007/978-3-319-70500-2_22. URL: https://doi.org/10.1007/978-3-319-70500-2%5C_22.

[110] Eunkyung Kim, Hyang-Sook Lee, and Jeongeun Park. “Towards Round-Optimal Secure Multiparty Computations: Multikey FHE Without a CRS”. In: *Information Security and Privacy - 23rd Australasian Conference, ACISP 2018, Wollongong, NSW, Australia, July 11-13, 2018, Proceedings*. Ed. by Willy Susilo and Guomin Yang. Vol. 10946. Lecture Notes in Computer Science. Springer, 2018, pp. 101–113. DOI: 10.1007/978-3-319-93638-3_7. URL: https://doi.org/10.1007/978-3-319-93638-3%5C_7.

[111] Christian Mouchet et al. “Multiparty Homomorphic Encryption from Ring-Learning-with-Errors”. In: *Proc. Priv. Enhancing Technol.* 2021.4 (2021), pp. 291–311. DOI: 10.2478/POPETS-2021-0071. URL: https://doi.org/10.2478/popets-2021-0071.

[112] Hyesun Kwak et al. “A General Framework of Homomorphic Encryption for Multiple Parties with Non-interactive Key-Aggregation”. In: *Applied Cryptography and Network Security - 22nd International Conference, ACNS 2024, Abu Dhabi, United Arab Emirates, March 5-8, 2024, Proceedings, Part II*. Ed. by Christina Pöpper and Lejla Batina. Vol. 14584. Lecture Notes in Computer Science. Springer, 2024, pp. 403–430. DOI: 10.1007/978-3-031-54773-7_16. URL: https://doi.org/10.1007/978-3-031-54773-7%5C_16.

[113] Nigel P. Smart. “Practical and Efficient FHE-Based MPC”. In: *Cryptography and Coding - 19th IMA International Conference, IMACC 2023, London, UK, December 12-14, 2023, Proceedings*. Ed. by Elizabeth A. Quaglia. Vol. 14421. Lecture Notes in Computer Science. Springer, 2023, pp. 263–283. DOI: 10.1007/978-3-031-47818-5_14. URL: https://doi.org/10.1007/978-3-031-47818-5%5C_14.

[114] Sylvain Chatel et al. “PELTA - Shielding Multiparty-FHE against Malicious Adversaries”. In: *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*. Ed. by Weizhi Meng et al. ACM, 2023, pp. 711–725. DOI: 10.1145/3576915.3623139. URL: https://doi.org/10.1145/3576915.3623139.

[115] Alexandre Bois et al. “Flexible and Efficient Verifiable Computation on Encrypted Data”. In: *Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10-13, 2021, Proceedings, Part II*. Ed. by Juan A. Garay. Vol. 12711. Lecture Notes in Computer Science. Springer, 2021, pp. 528–558. DOI: 10.1007/978-3-030-75248-4_19. URL: https://doi.org/10.1007/978-3-030-75248-4%5C_19.

[116] Radhika Garg et al. “Scalable Mixed-Mode MPC”. In: *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024*. IEEE, 2024, pp. 523–541. DOI: 10.1109/SP54263.2024.00106. URL: https://doi.org/10.1109/SP54263.2024.00106.

[117] Vivian Fang et al. “CostCO: An automatic cost modeling framework for secure multi-party computation”. In: *7th IEEE European Symposium on Security and Privacy, EuroS&P 2022, Genoa, Italy, June 6-10, 2022*. IEEE, 2022, pp. 140–153. DOI: 10.1109/EUROSP53844.2022.00017. URL: https://doi.org/10.1109/EuroSP53844.2022.00017.

[118] Daniel Demmler, Thomas Schneider, and Michael Zohner. “ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation”. In: *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2014*. 2015. URL: http://www.internetsociety.org/doc/aby---framework-efficient-mixed-protocol-secure-two-party-computation.

[119] Arpita Patra et al. “ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation”. In: *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*. Ed. by Michael D. Bailey and Rachel Greenstadt. USENIX Association, 2021, pp. 2165–2182. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/patra.

[120] Payman Mohassel and Peter Rindal. “ABY^3: A Mixed Protocol Framework for Machine Learning”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. Ed. by David Lie et al. ACM, 2018, pp. 35–52. DOI: 10.1145/3243734.3243760. URL: https://doi.org/10.1145/3243734.3243760.

[121] Wilko Henecka et al. “TASTY: Tool for Automating Secure Two-party Computations”. In: *Proceedings of the 17th ACM Conference on Computer and Communications Security*. CCS ’10. Chicago, Illinois, USA: ACM, 2010, pp. 451–462. ISBN: 978-1-4503-0245-6. DOI: 10.1145/1866307.1866358. URL: http://doi.acm.org/10.1145/1866307.1866358.

[122] Deepika Natarajan et al. “Chex-Mix: Combining Homomorphic Encryption with Trusted Execution Environments for Oblivious Inference in the Cloud”. In: *8th IEEE European Symposium on Security and Privacy, EuroS&P 2023, Delft, Netherlands, July 3-7, 2023*. IEEE, 2023, pp. 73–91. DOI: 10.1109/EUROSP57164.2023.00014. URL: https://doi.org/10.1109/EuroSP57164.2023.00014.

[123] Hanjun Li and Tianren Liu. “How to Garble Mixed Circuits that Combine Boolean and Arithmetic Computations”. In: *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part VI*. Ed. by Marc Joye and Gregor Leander. Vol. 14656. Lecture Notes in Computer Science. Springer, 2024, pp. 331–360. DOI: 10.1007/978-3-031-58751-1_12. URL: https://doi.org/10.1007/978-3-031-58751-1%5C_12.

[124] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. “How to Garble Arithmetic Circuits”. In: *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*. Ed. by Rafail Ostrovsky. IEEE Computer Society, 2011, pp. 120–129. DOI: 10.1109/FOCS.2011.40. URL: https://doi.org/10.1109/FOCS.2011.40.

[125] Banashri Karmakar et al. "Asterisk: Super-fast MPC with a Friend". In: *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024*. IEEE, 2024, pp. 542–560. DOI: 10.1109/SP54263.2024.00128. URL: https://doi.org/10.1109/SP54263.2024.00128.

[126] Sharemind MPC Team. *Integration of Sharemind MPC into Carbyne Stack*. Tech. rep. D-2-502. https://cyber.ee/uploads/Sharemind_MPC_CS_integration_a01ca476a7.pdf: Cybernetica AS, 2022.

[127] Martin Franz et al. "CBMC-GC: An ANSI C Compiler for Secure Two-Party Computations". In: *Compiler Construction - 23rd International Conference, CC 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*. Ed. by Albert Cohen. Vol. 8409. Lecture Notes in Computer Science. Springer, 2014, pp. 244–249. DOI: 10.1007/978-3-642-54807-9_15. URL: https://doi.org/10.1007/978-3-642-54807-9%5C_15.

[128] Edward Chen et al. "Silph: A Framework for Scalable and Accurate Generation of Hybrid MPC Protocols". In: *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*. IEEE, 2023, pp. 848–863. DOI: 10.1109/SP46215.2023.10179397. URL: https://doi.org/10.1109/SP46215.2023.10179397.

[129] Benjamin Levy et al. "COMBINE: COMpilation and Backend-INdependent vEctorization for Multi-Party Computation". In: *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*. Ed. by Weizhi Meng et al. ACM, 2023, pp. 2531–2545. DOI: 10.1145/3576915.3623181. URL: https://doi.org/10.1145/3576915.3623181.

[130] Nikolaj Volgushev et al. "Conclave: secure multi-party computation on big data". In: *Proceedings of the Fourteenth EuroSys Conference 2019, Dresden, Germany, March 25-28, 2019*. Ed. by George Candea, Robbert van Renesse, and Christof Fetzer. ACM, 2019, 3:1–3:18. DOI: 10.1145/3302424.3303982. URL: https://doi.org/10.1145/3302424.3303982.

[131] Henry Corrigan-Gibbs and Dan Boneh. "Prio: Private, Robust, and Scalable Computation of Aggregate Statistics". In: *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*. Ed. by Aditya Akella and Jon Howell. USENIX Association, 2017, pp. 259–282. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/corrigan-gibbs.

[132] Jennie Rogers et al. *VaultDB: A Real-World Pilot of Secure Multi-Party Computation within a Clinical Research Network*. 2022. arXiv: 2203.00146 [cs.DB]. URL: https://arxiv.org/abs/2203.00146.

[133] Felix Nikolaus Wirth et al. "EasySMPC: a simple but powerful no-code tool for practical secure multiparty computation". In: *BMC Bioinform.* 23.1 (2022), p. 531. DOI: 10.1186/S12859-022-05044-8. URL: https://doi.org/10.1186/s12859-022-05044-8.

[134] Nishanth Chandran et al. "EzPC: Programmable and Efficient Secure Two-Party Computation for Machine Learning". In: *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*. IEEE, 2019, pp. 496–511. DOI: 10.1109/EUROSP.2019.00043. URL: https://doi.org/10.1109/EuroSP.2019.00043.

[135] Dahlia Malkhi et al. "Fairplay—a Secure Two-party Computation System". In: *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*. SSYM'04. San Diego, CA: USENIX Association, 2004, pp. 20–20. URL: http://dl.acm.org/citation.cfm?id=1251375.1251395.

[136] Assaf Ben-David, Noam Nisan, and Benny Pinkas. "FairplayMP: A System for Secure Multiparty Computation". In: *Proceedings of the 15th ACM Conference on Computer and Communications Security*. CCS '08. Alexandria, Virginia, USA: ACM, 2008, pp. 257–266. ISBN: 978-1-59593-810-7. DOI: 10.1145/1455770.1455804. URL: http://doi.acm.org/10.1145/1455770.1455804.

[137] Najwa Aaraj et al. "FANNG-MPC: Framework for Artificial Neural Networks and Generic MPC". In: *IACR Cryptol. ePrint Arch.* (2023), p. 1918. URL: https://eprint.iacr.org/2023/1918.

[138] Aditya Shastri et al. *Private Computation Framework 2.0*. https://research.facebook.com/publications/private-computation-framework-2-0/. 2022.

[139] Brian Knott et al. "CrypTen: Secure Multi-Party Computation Meets Machine Learning". In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. Ed. by Marc'Aurelio Ranzato et al. 2021, pp. 4961–4973. URL: https://proceedings.neurips.cc/paper/2021/hash/2754518221cfbc8d25c13a06a4cb8421-Abstract.html.

[140] Ivan Damgård et al. "Confidential Benchmarking Based on Multiparty Computation". In: *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*. Ed. by Jens Grossklags and Bart Preneel. Vol. 9603. Lecture Notes in Computer Science. Springer, 2016, pp. 169–187. DOI: 10.1007/978-3-662-54970-4_10. URL: https://doi.org/10.1007/978-3-662-54970-4%5C_10.

[141] Benjamin Mood et al. "Frigate: A Validated, Extensible, and Efficient Compiler and Interpreter for Secure Computation". In: *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*. IEEE, 2016, pp. 112–127. DOI: 10.1109/EUROSP.2016.20. URL: https://doi.org/10.1109/EuroSP.2016.20.

[142] Christian Mouchet et al. "Helium: Scalable MPC among Lightweight Participants and under Churn". In: *IACR Cryptol. ePrint Arch.* (2024), p. 194. URL: https://eprint.iacr.org/2024/194.

[143] Lennart Braun et al. "FUSE - Flexible File Format and Intermediate Representation for Secure Multi-Party Computation". In: *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security, ASIA CCS 2023, Melbourne, VIC, Australia, July 10-14, 2023*. Ed. by Joseph K. Liu et al. ACM, 2023, pp. 649–663. DOI: 10.1145/3579856.3590340. URL: https://doi.org/10.1145/3579856.3590340.

[144] Niklas Büscher et al. "HyCC: Compilation of Hybrid Protocols for Practical Secure Computation". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. Ed. by David Lie et al. ACM, 2018, pp. 847–861. DOI: 10.1145/3243734.3243786. URL: https://doi.org/10.1145/3243734.3243786.

[145] Andrei Lapets et al. "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 2018, Menlo Park and San Jose, CA, USA, June 20-22, 2018*. Ed. by Ellen W. Zegura. ACM, 2018, 48:1–48:5. DOI: 10.1145/3209811.3212701. URL: https://doi.org/10.1145/3209811.3212701.

[146] Sergiu Carpov et al. "Manticore: Efficient Framework for Scalable Secure Multiparty Computation Protocols". In: *IACR Cryptol. ePrint Arch.* (2021), p. 200. URL: https://eprint.iacr.org/2021/200.

[147] Mariya Georgieva Belorgey et al. “Manticore: A Framework for Efficient Multiparty Computation Supporting Real Number and Boolean Arithmetic”. In: *J. Cryptol.* 36.3 (2023), p. 31. DOI: 10.1007/S00145-023-09464-4. URL: https://doi.org/10.1007/s00145-023-09464-4.

[148] Lennart Braun et al. “MOTION - A Framework for Mixed-Protocol Multi-Party Computation”. In: *ACM Trans. Priv. Secur.* 25.2 (2022), 8:1–8:35. DOI: 10.1145/3490390. URL: https://doi.org/10.1145/3490390.

[149] A. Bruggemann et al. “Don’t Eject the Impostor: Fast Three-Party Computation With a Known Cheater”. In: *2024 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2024, pp. 503–522. DOI: 10.1109/SP54263.2024.00164. URL: https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00164.

[150] Marcel Keller. “MP-SPDZ: A Versatile Framework for Multi-Party Computation”. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020. DOI: 10.1145/3372297.3417872. URL: https://doi.org/10.1145/3372297.3417872.

[151] Jack Doerner, David Evans, and Abhi Shelat. “Secure Stable Matching at Scale”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by Edgar R. Weippl et al. ACM, 2016, pp. 1602–1613. DOI: 10.1145/2976749.2978373. URL: https://doi.org/10.1145/2976749.2978373.

[152] Chang Liu et al. “ObliVM: A Programming Framework for Secure Computation”. In: *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. IEEE Computer Society, 2015, pp. 359–376. DOI: 10.1109/SP.2015.29. URL: https://doi.org/10.1109/SP.2015.29.

[153] Ahmad Al Badawi et al. “OpenFHE: Open-Source Fully Homomorphic Encryption Library”. In: *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography, Los Angeles, CA, USA, 7 November 2022*. Ed. by Michael Brenner, Anamaria Costache, and Kurt Rohloff. ACM, 2022, pp. 53–63. DOI: 10.1145/3560827.3563379. URL: https://doi.org/10.1145/3560827.3563379.

[154] Shai Halevi and Victor Shoup. “Design and implementation of HElib: a homomorphic encryption library”. In: *IACR Cryptol. ePrint Arch.* (2020), p. 1481. URL: https://eprint.iacr.org/2020/1481.

[155] Yihua Zhang, Aaron Steele, and Marina Blanton. “PICCO: A General-purpose Compiler for Private Distributed Computation”. In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security*. CCS ’13. Berlin, Germany: ACM, 2013, pp. 813–826. ISBN: 978-1-4503-2477-9. DOI: 10.1145/2508859.2516752. URL: http://doi.acm.org/10.1145/2508859.2516752.

[156] Yihua Zhang, Marina Blanton, and Ghada Almashaqbeh. “Implementing Support for Pointers to Private Data in a General-Purpose Secure Multi-Party Compiler”. In: *ACM Trans. Priv. Secur.* 21.2 (2018), 6:1–6:34. DOI: 10.1145/3154600. URL: https://doi.org/10.1145/3154600.

[157] Sameer Wagh et al. “Falcon: Honest-Majority Maliciously Secure Framework for Private Deep Learning”. In: *Proc. Priv. Enhancing Technol.* 2021.1 (2021), pp. 188–208. DOI: 10.2478/POPETS-2021-0011. URL: https://doi.org/10.2478/popets-2021-0011.

[158] John Liagouris et al. “SECRECY: Secure collaborative analytics in untrusted clouds”. In: *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17-19, 2023*. Ed. by Mahesh Balakrishnan and Manya Ghobadi. USENIX Association, 2023, pp. 1031–1056. URL: https://www.usenix.org/conference/nsdi23/presentation/liagouris.

[159] Junming Ma et al. “SecretFlow-SPU: A Performant and User-Friendly Framework for Privacy-Preserving Machine Learning”. In: *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. Boston, MA: USENIX Association, July 2023, pp. 17–33. ISBN: 978-1-939133-35-9. URL: https://www.usenix.org/conference/atc23/presentation/ma.

[160] Haoqi Wu et al. “Ditto: Quantization-aware Secure Inference of Transformers upon MPC”. In: *Proceedings of the 41st International Conference on Machine Learning*. Ed. by Ruslan Salakhutdinov et al. Vol. 235. Proceedings of Machine Learning Research. PMLR, July 2024, pp. 53346–53365. URL: https://proceedings.mlr.press/v235/wu24d.html.

[161] Zhicong Huang et al. “Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference”. In: *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*. Ed. by Kevin R. B. Butler and Kurt Thomas. USENIX Association, 2022, pp. 809–826. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/huang-zhicong.

[162] Robin William Hundt, Nora Khayata, and Thomas Schneider. “POSTER: SEEC — Memory Safety Meets Efficiency in Secure Two-Party Computation”. In: *ACSAC*. 2024.

[163] Haris Smajlović et al. “Sequre: a high-performance framework for secure multiparty computation enables biomedical data sharing”. In: *Genome Biology* 24.1 (2023), p. 5.

[164] Rishabh Poddar et al. “Senate: A Maliciously-Secure MPC Platform for Collaborative Analytics”. In: *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*. Ed. by Michael D. Bailey and Rachel Greenstadt. USENIX Association, 2021, pp. 2129–2146. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/poddar.

[165] Martin Burkhart et al. “SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics”. In: *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*. USENIX Association, 2010, pp. 223–240. URL: http://www.usenix.org/events/sec10/tech/full%5C_papers/Burkhart.pdf.

[166] Alfonso Iacovazzi et al. “Elementary secure-multiparty computation for massive-scale collaborative network monitoring: A quantitative assessment”. In: *Comput. Networks* 57.17 (2013), pp. 3728–3742. DOI: 10.1016/J.COMNET.2013.08.017. URL: https://doi.org/10.1016/j.comnet.2013.08.017.

[167] David W. Archer et al. *From Keys to Databases – Real-World Applications of Secure Multi-Party Computation*. 2018. DOI: 10.1093/comjnl/bxy090. URL: http://dx.doi.org/10.1093/comjnl/bxy090.

[168] Dan Bogdanov et al. “High-performance secure multi-party computation for data mining applications”. In: *Int. J. Inf. Sec.* 11.6 (2012), pp. 403–418. DOI: 10.1007/s10207-012-0177-2. URL: https://doi.org/10.1007/s10207-012-0177-2.

[169] Liina Kamm. “Privacy-preserving statistical analysis using secure multi-party computation”. PhD thesis. University of Tartu, 2015. URL: http://hdl.handle.net/10062/45343.

[170] Toomas Krips. “Improving performance of secure real-number operations”. https://dspace.ut.ee/handle/10062/63763. PhD thesis. University of Tartu, 2019.

[171] Sven Laur, Riivo Talviste, and Jan Willemson. “From Oblivious AES to Efficient and Secure Database Join in the Multiparty Setting”. In: *Applied Cryptography and Network Security - 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings*. Ed. by Michael J. Jacobson Jr. et al. Vol. 7954. Lecture Notes in Computer Science. Springer, 2013, pp. 84–101. ISBN: 978-3-642-38979-5. DOI: 10.1007/978-3-642-38980-1_6. URL: https://doi.org/10.1007/978-3-642-38980-1%5C_6.

[172] Sven Laur, Jan Willemson, and Bingsheng Zhang. “Round-Efficient Oblivious Database Manipulation”. In: *Information Security, 14th International Conference, ISC 2011, Xi’an, China, October 26-29, 2011. Proceedings*. Ed. by Xuejia Lai, Jianying Zhou, and Hui Li. Vol. 7001. Lecture Notes in Computer Science. Springer, 2011, pp. 262–277. ISBN: 978-3-642-24860-3. DOI: 10.1007/978-3-642-24861-0_18. URL: https://doi.org/10.1007/978-3-642-24861-0%5C_18.

[173] Peeter Laud. “Parallel Oblivious Array Access for Secure Multiparty Computation and Privacy-Preserving Minimum Spanning Trees”. In: *PoPETs* 2015.2 (2015), pp. 188–205. DOI: 10.1515/popets-2015-0011. URL: https://doi.org/10.1515/popets-2015-0011.

[174] Peeter Laud and Jaak Randmets. “A Domain-Specific Language for Low-Level Secure Multiparty Computation Protocols”. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*. ACM, 2015, pp. 1492–1503. DOI: 10.1145/2810103.2813664. URL: http://doi.acm.org/10.1145/2810103.2813664.

[175] Jaak Randmets. “Programming Languages for Secure Multi-party Computation Application Development”. http://hdl.handle.net/10062/56298. PhD thesis. University of Tartu, 2017.

[176] Dan Bogdanov et al. “Rmind: A Tool for Cryptographically Secure Statistical Analysis”. In: *IEEE Transactions on Dependable and Secure Computing* 15.3 (2018). http://doi.org/10.1109/TDSC.2016.2587623, pp. 481–495. DOI: 10.1109/TDSC.2016.2587623.

[177] Dan Bogdanov, Riivo Talviste, and Jan Willemson. “Deploying Secure Multi-Party Computation for Financial Data Analysis - (Short Paper)”. In: *Financial Cryptography and Data Security - 16th International Conference, FC 2012, Kralendijk, Bonaire, February 27 - March 2, 2012, Revised Selected Papers*. Ed. by Angelos D. Keromytis. Vol. 7397. Lecture Notes in Computer Science. Springer, 2012, pp. 57–64. DOI: 10.1007/978-3-642-32946-3_5. URL: https://doi.org/10.1007/978-3-642-32946-3%5C_5.

[178] Dan Bogdanov et al. “How the Estonian Tax and Customs Board Evaluated a Tax Fraud Detection System Based on Secure Multi-party Computation”. In: *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*. Ed. by Rainer Böhme and Tatsuaki Okamoto. Vol. 8975. Lecture Notes in Computer Science. Springer, 2015, pp. 227–234. DOI: 10.1007/978-3-662-47854-7_14. URL: https://doi.org/10.1007/978-3-662-47854-7%5C_14.

[179] Hendrik Ballhausen et al. “Privacy-friendly evaluation of patient data with secure multiparty computation in a European pilot study”. In: *npj Digital Medicine* 7.280 (2024). DOI: https://doi.org/10.1038/s41746-024-01293-4.

[180] David Archer et al. “Sharing sensitive department of education data across organizational boundaries using secure multiparty computation”. In: *Galois, Inc. and Georgetown University, Tech. Rep* 5 (2021).

[181] Ian Sweet et al. "Symphony: Expressive Secure Multiparty Computation with Coordination". In: *Art Sci. Eng. Program.* 7.3 (2023). DOI: 10.22152/PROGRAMMING-JOURNAL.ORG/2023/7/14. URL: https://doi.org/10.22152/programming-journal.org/2023/7/14.

[182] Sameer Wagh, Divya Gupta, and Nishanth Chandran. "SecureNN: 3-Party Secure Computation for Neural Network Training". In: *Proc. Priv. Enhancing Technol.* 2019.3 (2019), pp. 26–49. DOI: 10.2478/POPETS-2019-0035. URL: https://doi.org/10.2478/popets-2019-0035.

[183] Ebrahim M. Songhori et al. "TinyGarble: Highly Compressed and Scalable Sequential Garbled Circuits". In: *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. IEEE Computer Society, 2015, pp. 411–428. DOI: 10.1109/SP.2015.32. URL: https://doi.org/10.1109/SP.2015.32.

[184] Marie Beth van Egmond et al. "Privacy-preserving Anti-Money Laundering using Secure Multi-Party Computation". In: *IACR Cryptol. ePrint Arch.* (2024), p. 65. URL: https://eprint.iacr.org/2024/065.

[185] Ivan Damgård et al. "Asynchronous Multiparty Computation: Theory and Implementation". In: *Proceedings of the 12th International Conference on Practice and Theory in Public Key Cryptography: PKC '09*. Irvine. CA: Springer-Verlag, 2009, pp. 160–179. ISBN: 978-3-642-00467-4. DOI: 10.1007/978-3-642-00468-1_10. URL: http://dx.doi.org/10.1007/978-3-642-00468-1_10.

[186] Aseem Rastogi, Matthew A. Hammer, and Michael Hicks. "Wysteria: A Programming Language for Generic, Mixed-Mode Multiparty Computations". In: *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*. IEEE Computer Society, 2014, pp. 655–670. DOI: 10.1109/SP.2014.48. URL: https://doi.org/10.1109/SP.2014.48.

[187] Aseem Rastogi, Nikhil Swamy, and Michael Hicks. "WYS*: A Verified Language Extension for Secure Multi-party Computations". In: *CoRR* abs/1711.06467 (2017). arXiv: 1711.06467. URL: http://arxiv.org/abs/1711.06467.

[188] Marcella Hastings et al. "SoK: General Purpose Compilers for Secure Multi-Party Computation". In: *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 1220–1237. DOI: 10.1109/SP.2019.00028. URL: https://doi.org/10.1109/SP.2019.00028.

[189] Thomas Lorünser and Florian Wohner. "Performance Comparison of Two Generic MPC-frameworks with Symmetric Ciphers". In: *Proceedings of the 17th International Joint Conference on e-Business and Telecommunications, ICETE 2020 - Volume 2: SECRYPT, Lieusaint, Paris, France, July 8-10, 2020*. Ed. by Pierangela Samarati et al. ScitePress, 2020, pp. 587–594. DOI: 10.5220/0009831705870594. URL: https://doi.org/10.5220/0009831705870594.

[190] Fatih Aykurt. "ANALYSIS OF TWO VERSATILE MPC FRAMEWORKS MP-SPDZ AND MPYC". MA thesis. Middle East Technical University, 2023.

[191] Alexander Viand, Patrick Jattke, and Anwar Hithnawi. "SoK: Fully Homomorphic Encryption Compilers". In: *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 2021, pp. 1092–1108. DOI: 10.1109/SP40001.2021.00068. URL: https://doi.org/10.1109/SP40001.2021.00068.

[192] Charles Gouert, Dimitris Mouris, and Nektarios Georgios Tsoutsos. “SoK: New Insights into Fully Homomorphic Encryption Libraries via Standardized Benchmarks”. In: *Proc. Priv. Enhancing Technol.* 2023.3 (2023), pp. 154–172. DOI: 10.56553/POPETS-2023-0075. URL: https://doi.org/10.56553/popets-2023-0075.

[193] Dengguo Feng and Kang Yang. “Concretely efficient secure multi-party computation protocols: survey and more”. In: *Security and Safety* 1 (2022), p. 2021001.

[194] Yichen Li et al. “Metamorphic Testing of Secure Multi-party Computation (MPC) Compilers”. In: *Proc. ACM Softw. Eng.* 1.FSE (2024), pp. 1216–1237. DOI: 10.1145/3643781. URL: https://doi.org/10.1145/3643781.

[195] Qi Pang, Yuanyuan Yuan, and Shuai Wang. “MPCDiff: Testing and Repairing MPC-Hardened Deep Learning Models”. In: *31st Annual Network and Distributed System Security Symposium, NDSS 2024, San Diego, California, USA, February 26 - March 1, 2024*. The Internet Society, 2024. URL: https://www.ndss-symposium.org/ndss-paper/mpcdiff-testing-and-repairing-mpc-hardened-deep-learning-models/.

[196] United Nations Committee of Experts on Big Data and Data Science for Official Statistics. *UN Guide on Privacy-Enhancing Technologies for Official Statistics*. https://unstats.un.org/bigdata/task-teams/privacy/guide/. 2023.

[197] Pepijn Groen et al. “Privacy Enhancing Technologies Whitepaper: Developed by Centre of Excellence–Data Sharing and Cloud”. In: (2023).

[198] Marie Beth van Egmond et al. “Privacy-preserving Anti-Money Laundering using Secure Multi-Party Computation”. In: *Cryptology ePrint Archive* (2024).

[199] Meril Vaht. “The Analysis and Design of a Privacy-Preserving Survey System”. MA thesis. Institute of Computer Science, University of Tartu, 2015.

[200] Victoria Song. *Apple needs to explain that bug that resurfaced deleted photos*. https://www.theverge.com/2024/5/20/24161152/apple-ios-17-photo-bug. [Accessed 13-09-2024]. May 2024.

[201] Kevin Beaumont. *Recall: Stealing everything you’ve ever typed or viewed on your own Windows PC is now possible.* https://doublepulsar.com/recall-stealing-everything-youve-ever-typed-or-viewed-on-your-own-windows-pc-is-now-possible-da3e12e9465e. [Accessed 13-09-2024].

[202] Cyber Safety Review Board. *Review of the Summer 2023 Microsoft Exchange Online Intrusion*. https://www.cisa.gov/resources-tools/resources/CSRB-Review-Summer-2023-MEO-Intrusion. Mar. 2023.

[203] H. Birkholz et al. *Remote ATtestation procedureS (RATS) Architecture*. RFC 9334. RFC Editor, Jan. 2023. URL: https://www.rfc-editor.org/rfc/rfc9334.txt.

[204] Hugo Krawczyk. “SIGMA: The ‘SIGn-and-MAc’ Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols”. In: *Advances in Cryptology - CRYPTO 2003*. Ed. by Dan Boneh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 400–425.

[205] Samira Briongos et al. “No Forking Way: Detecting Cloning Attacks on Intel SGX Applications”. In: *Proceedings of the 39th Annual Computer Security Applications Conference*. ACSAC ’23. Austin, TX, USA: Association for Computing Machinery, 2023, pp. 744–758. ISBN: 9798400708862. DOI: 10.1145/3627106.3627187. URL: https://doi.org/10.1145/3627106.3627187.

[206] Stephan van Schaik et al. “SoK: SGX.Fail: How Stuff Get eXposed”. In: 2022.

[207] Antonio Muñoz et al. “A survey on the (in)security of trusted execution environments”. In: *Computers & Security* 129 (2023), p. 103180. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2023.103180. URL: https://www.sciencedirect.com/science/article/pii/S0167404823000901.

[208] Jaak Randmets and Armin Kisand. *An Overview of Vulnerabilities and Mitigations of Intel SGX Applications*. Technical Report D-2-116 v1.3. 2024. Institute of Information Security, Cybernetica AS, 2024. URL: https://cyber.ee/uploads/D_2_116_An_Overview_of_Vulnerabilities_and_Mitigations_of_Intel_SGX_Applications_c1282b1505.pdf.

[209] Mengyuan Li et al. “SoK: Understanding Design Choices and Pitfalls of Trusted Execution Environments”. In: *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*. 2024, pp. 1600–1616.

[210] Douglas Everson, Long Cheng, and Zhenkai Zhang. “Log4shell: Redefining the web attack surface”. In: *Proc. Workshop Meas., Attacks, Defenses Web (MADWeb)*. 2022, pp. 1–8.

[211] Zitai Chen et al. “VoltPillager: Hardware-based fault injection attacks against Intel SGX Enclaves using the SVID voltage scaling interface”. In: *30th USENIX Security Symposium (USENIX Security 21)*. Vancouver, B.C.: USENIX Association, Aug. 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/chen-zitai.

[212] Aakash Gangolli, Qusay H. Mahmoud, and Akramul Azim. “A Systematic Review of Fault Injection Attacks on IoT Systems”. In: *Electronics* 11.13 (2022). ISSN: 2079-9292. DOI: 10.3390/electronics11132023. URL: https://www.mdpi.com/2079-9292/11/13/2023.

[213] Microsoft. *Physical security of Azure datacenters - Microsoft Azure*. https://learn.microsoft.com/en-us/azure/security/fundamentals/infrastructure. [Accessed 08-10-2024].

[214] AWS. *Data Centers - Our Controls*. https://aws.amazon.com/compliance/data-center/controls/. [Accessed 08-10-2024].

[215] Google. *How Google protects the physical-to-logical space in a data center*. https://www.google.com/about/datacenters/data-security/. [Accessed 08-10-2024].

[216] Intel®. *Intel® Software Guard Extensions Trusted Computing Base Recovery*. White Paper. 2019. URL: https://cdrdv2-public.intel.com/740676/Intel-SGX-Trusted-Computing-Base-Recovery-2019.pdf.

[217] Intel®. *Intel® Software Guard Extensions (Intel® SGX) Data Center Attestation Primitives: ECDSA Quote Library API*. Documentation. Aug. 2023. URL: https://download.01.org/intel-sgx/sgx-dcap/1.21/linux/docs/Intel_SGX_ECDSA_QuoteLibReference_DCAP_API.pdf.

[218] AWS. *The Security Design of the AWS Nitro System*. White Paper. 2024. URL: https://docs.aws.amazon.com/pdfs/whitepapers/latest/security-design-of-aws-nitro-system/security-design-of-aws-nitro-system.pdf.

[219] AMD. *SEV Secure Nested Paging Firmware ABI Specification*. Specification. Sept. 2023. URL: https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/specifications/56860.pdf.

[220] Robert Buhren, Christian Werling, and Jean-Pierre Seifert. “Insecure until proven updated: analyzing AMD SEV’s remote attestation”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019, pp. 1087–1099.

[221] Jo Van Bulck et al. “LVI: Hijacking transient execution through microarchitectural load value injection”. In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2020, pp. 54–72.

[222] Intel®. *Load Value Injection*. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/load-value-injection.html. [Accessed 14-09-2024].

[223] Mathias Morbitzer et al. “Severity: Code injection attacks against encrypted virtual machines”. In: *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE. 2021, pp. 444–455.

[224] Moritz Schneider et al. *SoK: Hardware-supported Trusted Execution Environments*. 2022. arXiv: 2205.12742 [cs.CR]. URL: https://arxiv.org/abs/2205.12742.

[225] Ayaz Akram et al. “Sok: Limitations of confidential computing via tees for high-performance compute systems”. In: *2022 IEEE International Symposium on Secure and Private Execution Environment Design (SEED)*. IEEE. 2022, pp. 121–132.

[226] Apple. *Secure Enclave*. https://support.apple.com/guide/security/secure-enclave-sec59b0b31ff/web. [Accessed 01-10-2024].

[227] Sandro Pinto and Nuno Santos. “Demystifying arm trustzone: A comprehensive survey”. In: *ACM computing surveys (CSUR)* 51.6 (2019), pp. 1–36.

[228] IBM. *IBM Secure Execution for Linux*. Solution Brief. 2022. URL: https://www.ibm.com/downloads/cas/O158MBWG.

[229] C. Bornträger et al. “Secure your cloud workloads with IBM Secure Execution for Linux on IBM z15 and LinuxONE III”. In: *IBM Journal of Research and Development* 64.5/6 (2020), 2:1–2:11. DOI: 10.1147/JRD.2020.3008109.

[230] IBM. *Introducing IBM Secure Execution for Linux describe secure execution concepts, how to set it up as a cloud provider, and how to secure your workload as a workload owner.* https://www.ibm.com/docs/en/linux-on-systems?topic=management-secure-execution. [Accessed 21-10-2024].

[231] Guerney D. H. Hunt et al. “Confidential computing for OpenPOWER”. In: *Proceedings of the Sixteenth European Conference on Computer Systems*. EuroSys ’21. Online Event, United Kingdom: Association for Computing Machinery, 2021, pp. 294–310. ISBN: 9781450383349. DOI: 10.1145/3447786.3456243. URL: https://doi.org/10.1145/3447786.3456243.

[232] Intel®. *Intel® Software Guard Extensions (Intel® SGX)*. https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/software-guard-extensions.html. [Accessed 01-10-2024].

[233] Intel®. https://download.01.org/intel-sgx/latest/linux-latest/docs/. [Accessed 01-10-2024].

[234] Victor Costan. “Intel SGX explained”. In: *IACR Cryptol, EPrint Arch* (2016).

[235] Nico Weichbrodt et al. “AsyncShock: Exploiting synchronisation bugs in Intel SGX enclaves”. In: *Computer Security–ESORICS 2016: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part I 21*. Springer. 2016, pp. 440–457.

[236] Scott Constable et al. “{AEX-Notify}: Thwarting Precise {Single-Stepping} Attacks through Interrupt Awareness for Intel {SGX} Enclaves”. In: *32nd USENIX Security Symposium (USENIX Security 23)*. 2023, pp. 4051–4068.

[237] Intel®. *Intel® Software Guard Extensions (Intel® SGX) SDK for Linux* OS. Developer Reference*. Documentation. Sept. 2024. URL: https://download.01.org/intel-sgx/sgx-linux/2.25/docs/Intel_SGX_Developer_Reference_Linux_2.25_Open_Source.pdf.

[238] Intel®. *Intel® Trust Domain Extensions*. White Paper. Feb. 2023. URL: https://cdrdv2.intel.com/v1/dl/getContent/690419.

[239] Intel®. *Intel® Trust Domain Extensions (Intel® TDX)*. https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/overview.html. [Accessed 01-10-2024].

[240] Pau-Chen Cheng et al. "Intel tdx demystified: A top-down approach". In: *ACM Computing Surveys* 56.9 (2024), pp. 1–33.

[241] Erdem Aktas et al. *Intel trust domain extensions (TDX) security review*. Technical Report. Google, 2023.

[242] Magnus Kulke. *Building Trust into OS images for Confidential Containers*. 2024. URL: https://confidentialcontainers.org/blog/2024/03/01/building-trust-into-os-images-for-confidential-containers/ (visited on 08/01/2024).

[243] Vikram Narayanan et al. "Remote attestation of confidential VMs using ephemeral vTPMs". In: *Proceedings of the 39th Annual Computer Security Applications Conference*. ACSAC '23. Austin, TX, USA: Association for Computing Machinery, 2023, pp. 732–743. DOI: 10.1145/3627106.3627112. URL: https://doi.org/10.1145/3627106.3627112.

[244] AMD. *Strengthening VM isolation with integrity protection and more*. White Paper. Jan. 2020. URL: https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf.

[245] Petar Paradžik, Ante Derek, and Marko Horvat. "Formal Security Analysis of the AMD SEV-SNP Software Interface". In: *arXiv preprint arXiv:2403.10296* (2024).

[246] Charles Garcia-Tobin and Mark Knight. "Elevating Security with Arm CCA: Attestation and verification are integral to adopting confidential computing." In: *Queue* 22.2 (2024), pp. 39–56.

[247] Xupeng Li et al. "Design and verification of the arm confidential compute architecture". In: *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 2022, pp. 465–484.

[248] John Redford and Xiang Wen Kuan. *AWS Nitro System API & Security Claims*. Technical Report. NCC Group Security Services, Inc., Apr. 2022. URL: https://www.nccgroup.com/us/research-blog/public-report-aws-nitro-system-api-security-claims/.

[249] Paweł Płatek. *A few notes on AWS Nitro Enclaves: Attack surface*. https://blog.trailofbits.com/2024/09/24/notes-on-aws-nitro-enclaves-attack-surface/. [Accessed 08-10-2024]. Sept. 2024.

[250] AWS. *Data protection in Amazon EC2 - Amazon Elastic Compute Cloud*. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/data-protection.html. [Accessed 22-10-2024].

[251] Intel®. *Intel® SGX SDK*. https://github.com/intel/linux-sgx.

[252] Microsoft. *Open Enclave SDK*. https://openenclave.io/sdk/. [Accessed 01-10-2024].

[253] AWS. *AWS Nitro Enclaves SDK for C*. https://github.com/aws/aws-nitro-enclaves-sdk-c.

[254] Chia-Che Tsai, Donald E Porter, and Mona Vij. “Graphene-SGX: A practical library OS for unmodified applications on SGX”. In: *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. 2017, pp. 645–658.

[255] *Enarx*. https://enarx.dev/. [Accessed 01-10-2024].

[256] Mathias Brossard et al. *Private delegated computations using strong isolation*. Technical report. Systems Research Group, Arm Research, 2022, p. 20. DOI: https://doi.org/10.48550/arXiv.2205.03322.

[257] *KubeVirt CVM*. https://github.com/cc-api/kubevirt-cvm.

[258] Confidential Containers. *Confidential Containers*. https://confidentialcontainers.org/. [Accessed 09-09-2024].

[259] Lorenzo Fero, Antonio Lioy, et al. “Standard-Based Remote Attestation: The Veraison Project”. In: *Proceedings of The Italian Conference on Cybersecurity (ITASEC 2024)*. CEUR-WS. 2024, pp. 1–13.

[260] Gianluca Scopelliti, Christoph Baumann, and Jan Tobias Mühlberg. “Understanding Trust Relationships in Cloud-Based Confidential Computing”. In: *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE. 2024, pp. 169–176.

[261] Microsoft. *Azure confidential virtual machines FAQ*. https://learn.microsoft.com/en-us/azure/confidential-computing/confidential-vm-faq#can-i-perform-attestation-for-my-intel-based-confidential-vms-. [Accessed 01-10-2024].