

# A Survey of Secure Computation Using Trusted Execution Environments






XIAOGUO LI , Singapore Management University, Singapore  
 BOWEN ZHAO , Xidian University, China  
 GUOMIN YANG , Singapore Management University, Singapore  
 TAO XIANG , Chongqing University, China  
 JIAN WENG , Jinan University, China  
 ROBERT H. DENG , Singapore Management University, Singapore

As an essential technology underpinning trusted computing, the trusted execution environment (TEE) allows one to launch computation tasks on both on- and off-premises data while assuring confidentiality and integrity. This article provides a systematic review and comparison of TEE-based secure computation protocols. We first propose a taxonomy that classifies secure computation protocols into three major categories, namely secure outsourced computation, secure distributed computation and secure multi-party computation. To enable a fair comparison of these protocols, we also present comprehensive assessment criteria with respect to four aspects: setting, methodology, security and performance. Based on these criteria, we review, discuss and compare the state-of-the-art TEE-based secure computation protocols for both general-purpose computation functions and special-purpose ones, such as privacy-preserving machine learning and encrypted database queries. To the best of our knowledge, this article is the first survey to review TEE-based secure computation protocols and the comprehensive comparison can serve as a guideline for selecting suitable protocols for deployment in practice. Finally, we also discuss several future research directions and challenges.

CCS Concepts: • **Security and privacy** → **Tamper-proof and tamper-resistant designs; Trust frameworks; Privacy-preserving protocols; Management and querying of encrypted data.**




Additional Key Words and Phrases: trusted computing, trusted execution environment, security computation, federated learning

## ACM Reference Format:

Xiaoguo Li , Bowen Zhao , Guomin Yang , Tao Xiang , Jian Weng , and Robert H. Deng . 20XX. A Survey of Secure Computation Using Trusted Execution Environments. *ACM Comput. Surv.* 37, 4, Article 111 (August 20XX), 39 pages. <https://doi.org/XXXXXXX.XXXXXXX>

## 1 INTRODUCTION

The popularity of data-driven analytics has brought exponential data volume growth, especially in the Internet of Things (IoT) era. There will be approximately 572 zettabytes of data produced by 2030 [10], which is about 30 times that of today. Accompanying the rapid data growth is the continuously evolving computing technologies for performing complex analytical tasks over huge-scale data.

Authors' addresses: Xiaoguo Li , Singapore Management University, 81 Victoria St., Singapore, Singapore, 188065, [xiaoguoli@smu.edu.sg](mailto:xiaoguoli@smu.edu.sg); Bowen Zhao , Xidian University, Guangzhou, China, 510555, [zhaobw29@163.com](mailto:zhaobw29@163.com); Guomin Yang , Singapore Management University, Singapore, Singapore, 188065, [gmyang@smu.edu.sg](mailto:gmyang@smu.edu.sg); Tao Xiang , Chongqing University, Chongqing, China, 400044, [txiang@cqu.edu.cn](mailto:txiang@cqu.edu.cn); Jian Weng , Jinan University, Guangzhou, China, 510632, [cryptjweng@gmail.com](mailto:cryptjweng@gmail.com); Robert H. Deng , Singapore Management University, Singapore, Singapore, 188065, [robertdeng@smu.edu.sg](mailto:robertdeng@smu.edu.sg).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2018 Association for Computing Machinery.

0360-0300/20XX/8-ART111 \$15.00

<https://doi.org/XXXXXXX.XXXXXXX>

To eliminate the burden and cost of deploying and maintaining a high-performance computing infrastructure, many organizations resort to cloud services nowadays. Nonetheless, under such an off-premises computing model, a cloud server may extract valuable or sensitive information for its own benefit, which could damage the interests of the data owners. Moreover, in computation tasks involving multiple data sources, protecting the interest of each data owner is also an essential requirement. As a general terminology, secure computation refers to technologies that can ensure data privacy and/or integrity throughout the whole computation.

In general, the existing secure computation frameworks mainly target evaluating a function over encrypted or obfuscated inputs. Its actual form depends on the computation task and setting. For example, in a multi-party setting, secure computation allows multiple parties with private inputs to evaluate a joint function based on all private inputs, such as a statistical function in a voting scenario or a sorting function in an auction scenario. During the computation, it is necessary to ensure the privacy of each individual input as well as the correctness/integrity of the output. In the outsourced computation setting, secure computation allows a client (or clients) to outsource computation tasks, which could involve sensitive data, to an untrusted cloud server. The main challenge of secure computation is: *How can we securely and efficiently evaluate functions on private inputs in the presence of untrusted or malicious parties participating in the computation?*

**Cryptographic Approaches for Secure Computation.** In the past four decades, many cryptographic primitives and protocols have been proposed to address secure computation problems. Following Yao's garbled circuits (GC, [155]) and GMW's scheme [56] since the 1980s, enormous efforts have been put to improve security and efficiency in secure multi-party computation. For example, the cut-and-choose mechanism [88] extended Yao's protocol against malicious adversaries, and dual execution [76] improved GMW's scheme to achieve stronger security. Other examples of cryptographic primitives for secure computation include Oblivious RAM (ORAM, [57, 136]), homomorphic encryption (HE, [55]), and secret sharing (SS, [127]). ORAM-based approaches [46, 89, 148] build shared ORAM, and then data privacy is achieved by multiple rounds of interactions between the CPU and the shared memory. Since all these interactions are encrypted and the shared ORAM re-encrypts and shuffles the data whenever it is accessed, low efficiency becomes a major drawback. HE-based solutions [37, 38] allow computation on ciphertexts directly, however some of them (PHE [108], SWHE [37, 38]) cannot support arbitrary computation, whereas others (e.g., FHE [55]) are still impractical because of the extensive computation cost. SS-based solutions [36, 61] split a secret into multiple shares and then distribute them among two or more players. With the shares, one can reconstruct the original secret from them (e.g., by Lagrangian interpolation). In recent years, secret sharing has shown its superiority and serves as an essential component in several mixed frameworks for secure computation, such as ABY [42] and ABY2.0 [113].

Yet, the aforementioned cryptographic approaches are still not widely adopted due to their strong trust assumptions or extensive computation and communication overhead, especially when dealing with complex computation tasks or large datasets. Many approaches (e.g., [36, 38, 113, 143, 162]) assume the involved parties are semi-honest, meaning those parties, despite being curious about others' private inputs, would follow the protocol steps honestly. Such an assumption does not hold in many application domains where parties have incentive to gain more benefits or advantages over their competitors by deviating from the protocol specification. On the other hand, defending malicious parties using pure crypto-based solutions introduces a large overhead. We note that secure computation protocols that work in the semi-honest setting can be converted to defend malicious participants (e.g., [17, 78, 88]), but with extensive computation and communication costs.

**TEE-based Secure Computation.** Today's growing awareness of Trusted Execution Environment (TEE [115]) has attracted many interests to explore alternative ways to build practical general/special-purpose secure computation protocols. Thanks to the trusted hardware, TEE-based

secure computation protocols are shown to be more efficient than the traditional crypto-based approaches while offering both confidentiality and integrity guarantees. *Therefore, TEE serves as a promising technology to enable practical and scalable secure computation for real applications. This article reviews and compares the state-of-the-art TEE-based secure computation protocols and discusses the remaining challenges, open problems and future research directions.*

## 1.1 Background of Trusted Execution Environment

TEE provides an isolated environment (called secure *enclave*<sup>1</sup>) that safeguards processed data by encrypting the incoming and outgoing data. Furthermore, TEE provides mechanisms to ensure the computation is correctly executed with an integrity guarantee. More importantly, TEE protects the data and computation against any potentially malicious entity residing in the system (including the kernel, hypervisor, etc.). Thus, TEE-based secure computation has attracted numerous researchers from academia and industrial communities in the last two decades [2, 27, 35, 85, 92, 96] because of three appealing features.

- *General purpose computation.* The supported functions executed in TEE are usually denoted as user-defined programs; thus, it permits a broad spectrum of functions.
- *Integrity and privacy.* TEE provides guarantees of integrity as well as confidentiality. Confidentiality means the attackers cannot obtain sensitive information from the computation, and integrity ensures that the program executes correctly.
- *High performance.* TEE computes the user-defined programs by decrypting all encrypted inputs in enclaves and executing the program on plaintext. Therefore, it enjoys the CPU speed performance, except the overhead for handling the encrypted inputs.

According to the design, current TEE can be classified into two categories: (1) TEE without a secure counter just supports simple stateless functions (e.g., smart cards [117, 134]) and (2) TEE with a secure counter supports complex stateful functions. In recent years, many TEE designs focus on the latter such as the Trusted Platform Modules (TPM/vTPM) [138], Intel TXT [54], Intel SGX [34], ARM's TrustZone [86], Sanctum [35], KeyStone [83] and AMD SEV [71]. These designs vary significantly in terms of architectural choices, instruction sets, implementation details, cryptographic suites, as well as security features [112].

**1.1.1 Security Mechanisms.** TEE exhibits a broad range of security features [50] to achieve confidentiality and integrity of the computation. With these features, TEE has been adopted widely by complex secure systems [29]. We present several essential features related to secure computation below.

- *Secure boot [131].* When the host starts, secure boot loads immutable and verified images into an enclave to ensure a chain of trust among the enclave images, operating system components, and configurations. Before the operating system starts, a secure boot ensures the TEE is loaded correctly and no hosts, even the hypervisor, can tamper with it.
- *Attestation [2].* Attestation establishes trust between an enclave and another enclave or a remote machine by building a secure channel. Generally, there are two attestation mechanisms: local attestation and remote attestation. While two enclaves on the same platform can attest to each other using local attestation, remote attestation allows a client to authenticate that its program is configured and executed correctly (i.e., not modified or tampered with) in a remote server.
- *Isolated execution [96].* TEE loads user-defined programs into enclaves and then performs isolated execution for each program. During execution, sensitive data (including code and

<sup>1</sup>In the rest of this survey, we use the terms 'TEE' and 'enclave' interchangeably

data) is maintained in EPCs (Enclave Page Caches) within the PRM (Processor Reserved Memory), which is pre-configured in DRAM in bootstrapping phase. The code loaded inside an enclave can access both non-PRM and PRM memory. On the other hand, external programs cannot access any data in the PRM memory.

- *Sealing [2]*. Sealing is a mechanism to let a TEE securely persist and retrieve “secrets” on the local disk. Sealing binds the secrets to a specific enclave identity by a sealing key derived from the CPU hardware. Using the sealing key, TEE encrypts the enclave’s secrets and stores them on a local disk. Conversely, TEE retrieves the secrets from the disk by the unsealing mechanism.

**1.1.2 Vulnerabilities.** Despite its appealing features, TEE is also vulnerable to several attacks, and the survey in [50] discusses these attacks in details. Below we briefly review two major attacks that would affect secure computation. First, TEE may suffer from side-channel attacks which could breach data privacy.

- *Timing side channel [60]*. Given different inputs, the running time of the program in the enclave may be different too. By monitoring the difference over the execution time, an attacker may derive some sensitive information about the input, e.g., a secret value reflecting how many times a loop is executed. To solve this problem, one approach is to ensure the program loaded into the enclave always takes approximately the same amount of execution time.
- *Memory side channel [123]*. In some programs, the access pattern to the non-PRM memory may depend on the enclave’s secret. For example, if the enclave runs a binary search program, the access pattern is a path from the root node to a leaf node, which may reveal the real secret. Therefore the loaded program should remove any correlation between the memory access pattern and the insider secret.
- *Network side channel [164]*. In a distributed system, network access patterns (e.g., sorting or hash partitioning) may reveal sensitive information. Therefore, the network access protocol should also be well-designed such that no sensitive information can be deduced from the patterns.

Second, TEEs may also be vulnerable to rewind attacks (or replay attacks), which allows an attacker to actively reset the state of the computation [16]. Note that such attacks may result in catastrophic consequences for stateful computation, such as limited-attempt password checking [133]. We cannot always expect the TEE to maintain the state securely for two reasons. (1) TEE may be stateless in practice. (2) Even for stateful TEE, the limited protected memory makes it fail to correctly maintain a large state. Therefore, the designer should be careful and provides special mechanisms to counter rewind attacks.

## 1.2 Our Contributions

TEE provides a promising hardware-based alternative for achieving secure computation with a CPU-level performance, and thus bridges the gap between academic research and industrial adoption of secure computation. Trusted hardware provides a powerful abstraction for building secure systems and can be applied to many potential applications, such as machine learning algorithms [81, 97, 98, 102, 106, 142], cloud computing [15, 33, 94, 122, 126, 141], blockchain [18, 22, 87, 159], network traffic analysis [47], scientific computation [130], data analytics [41, 75, 118, 164, 165], and privacy-preserving COVID-19 contact tracing [73].

To help readers systemically perceive the core design of TEE-based secure computation protocols, this article presents a comprehensive survey on the state of the art of this research field. Our contributions are summarized as follows:

- First, we propose two taxonomies for TEE-based secure computation protocols from the computing framework and TEE utilization perspectives. According to the framework, we categorize secure computation protocols into secure outsourced computation (SOC), secure distributed computation (SDC), and secure multiparty computation (SMC). Based on the TEE utilization, the existing protocols are classified into three categories: (1) pure TEE based protocols, which use TEE as a black box, (2) TEE with oblivious primitives, which can prevent side-channel and leakage attacks, and (3) TEE with trusted ledger, which can resist attacks targeting TEE state.
- Second, comprehensive assessment criteria are presented for the purpose of evaluating and comparing the existing protocols. The assessment criteria are made from four dimensions: (1) *setting* that specifies the computing function and framework; (2) *methodology* that describes the TEE utilization method and the additional crypto-primitives used; (3) *security* that highlights the security features of a protocol, such as privacy, integrity, fairness, and freshness; and (4) *performance* that shows the experimental configurations and results.
- Third, comprehensive comparisons among the existing protocols are made in accordance with the above assessment criteria. Specifically, this article first compares the existing proposals for general-purpose secure computation. Then detailed comparisons are made on two specific computational tasks: (1) machine learning, which consists of training and inference; and (2) encrypted database queries.
- Lastly, we discuss several future research directions and challenges in the light of our observations on the state-of-the-art TEE-enabled secure computation solutions.

This survey also aims to help non-specialists comprehend TEE-enabled secure computation and assist practitioners to determine appropriate TEE technologies for system deployment.

## 2 TAXONOMY AND ASSESSMENT CRITERIA

This section presents taxonomy and assessment criteria for TEE-based secure computation in Sec. 2.1 and in Sec. 2.2, respectively.

### 2.1 Taxonomy of TEE-based Secure Computation

We categorize TEE-based secure computation protocols based on two orthogonal aspects: the computation framework and the TEE utilization.

Based on the computation framework, we separate TEE-based secure computation protocols into the following three categories:

- **Secure outsourced computation (SOC).** Fig. 1a shows the framework of SOC, in which data owners outsource encrypted data to a single cloud server. Subsequently, an authorized data user will run an SOC protocol with the cloud server to realize desired computation tasks. In practice, the single cloud is an untrusted entity, which may behave as a passive/semi-honest attacker or an active/malicious attacker.
- **Secure distributed computation (SDC).** Fig. 1b shows the framework of SDC, in which the encrypted data is outsourced to multiple servers. Specifically, one master node is in charge of coordinating the computational tasks, and several slave nodes execute the computation in a distributed manner. An SDC protocol is run between an authorized data user and these server nodes. Specifically, each slave node calculates an intermediate result, and the master node responds data user with the final output, which is reconstructed from intermediate results. The master node and slave nodes may be semi-honest or malicious. Compared to SOC, the interactions among the servers and the intermediate results in SDC expose a larger attack surface to attackers.

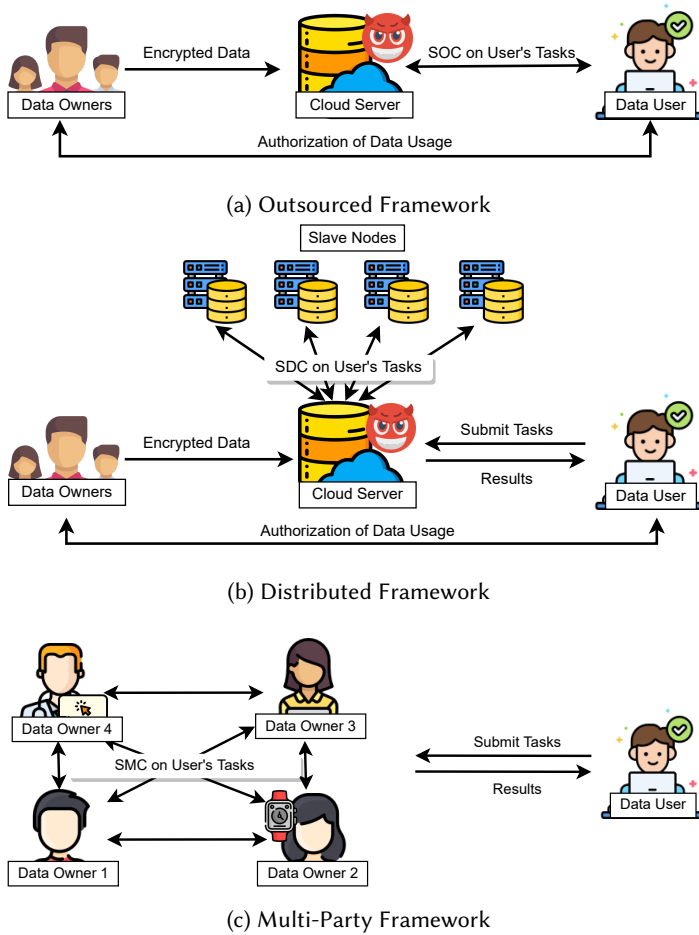


Fig. 1. Three Different Frameworks For Secure Computation

- **Secure multi-party computation (SMC).** Fig. 1c shows the framework of SMC, in which data owners jointly run a SMC protocol themselves without losing control of their data. The difference between SMC and SDC is that SMC does not involve a master node to coordinate computation tasks. The data owners (also called parties) participating in the computation don't trust each other. Specifically, each party in SMC tries to protect their sensitive information from other parties who may behave semi-honestly or even maliciously (i.e., by undermining the protocol).

Based on TEE utilization, i.e., whether it is used alone or in conjunction with other tools for preventing side-channel or state-related attacks, we can also categorize the protocols into the following types:

- **TEE as a black box.** TEE enables general-purpose computation with confidential and integrity guarantee against eavesdropping adversaries. Because the input and output are encrypted by an authenticated encryption scheme (e.g., AES-GCM), the untrusted host cannot learn sensitive information. Many existing solutions regard TEE as a black box and fall in

this category. If a protocol requires a stateful hardware device, we mark it as “stateful TEE”; otherwise, we mark it as “stateless TEE”.

- **TEE with oblivious primitives.** As mentioned earlier, although a host cannot directly access TEE’s state, sensitive information may still be leaked via different side-channels. To prevent such leakage, well-designed oblivious primitives (OP) [124] and differential obliviousness [1, 95, 153] are leveraged to mitigate side-channel attacks from different levels. The resulting proposals, such as ZeroTrace [123] and ObliJoin[28], not only achieve confidentiality and integrity but also avoid sensitive information leakage from other channels. The protocols in this category are labeled as “stateful TEE + OP” or “stateless TEE + OP” for comparison.
- **TEE with trusted public ledger.** When TEE is deployed on a remote malicious host, the latter may submit illegal inputs to the enclave. For example rewind attack in a stateful computation, the host may provide a stale encrypted state to the enclave, which may lead to catastrophic effects. Therefore a trusted public ledger (PL) [72] can be introduced to ensure the legitimacy of inputted state, and the protocols that fall in this category are labeled as “stateful TEE + PL” or “stateless TEE + PL” for comparison.

## 2.2 Assessment Criteria

For a systematic and fair comparison of existing proposals for TEE-based secure computation, we present the common assessment criteria with respect to four aspects: setting, methodology, security and performance (shown in Fig. 2). These criteria are applied in the comparison of existing protocols throughout this survey.

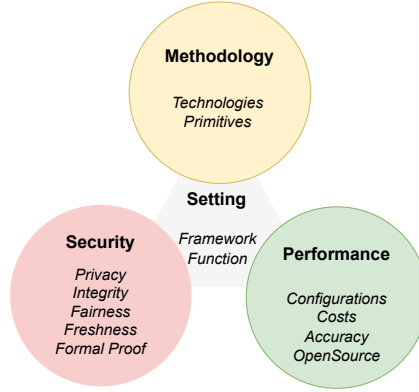


Fig. 2. Assessment Criteria

**2.2.1 Setting.** Secure computation protocols are designed for different purposes and can follow different computation frameworks. For achieving a fair comparison, we distinguish the secure computation setting from two aspects: *framework* and *function* as shown in Fig. 2. The framework specifies the architecture of the designed protocol, which has been elaborated in Sec. 2.1. The function represents the primary goal of the protocol, such as a machine learning task (training or inference), scientific computation (such as matrix multiplication and polynomial evaluation), data analytics (such as SQL queries and statistical analysis), or a general-purpose function.

**2.2.2 Methodology.** Methodology describes the design principle of a proposal and will guide developers to facilitate the practical deployment. We describe the methodology of TEE-based protocols from two aspects (shown in Fig. 2): TEE *utilization* and crypto *primitives*. The former

describes how TEE is used in a protocol (Sec. 2.1) and the latter presents which cryptographic toolkit is leveraged. It is worth noting that many crypto primitives are employed to deal with the security issues outside the TEE, such as authenticated encryption, digital signatures, secret sharing, and zero-knowledge proof. Therefore, we also highlight crypto primitives in the comparison.

**2.2.3 Security.** Generally, security evaluation consists of two dimensions: *security goals* and *adversarial models*. According to system requirements, different security goals are defined under different adversarial models. We present the common security goals and adversarial models as follows.

- **Security goals.** Security goals define which security requirements should be achieved in a secure computation protocol. In this article, four types of security goals are considered: privacy, integrity, fairness, and freshness. Specifically,
  - *Privacy.* Privacy in secure computation means that sensitive data (including input, intermediate result, output, or other metadata during computation such as access pattern) is not visible to potential attackers. Moreover, the sensitive data may include the concrete queries in privacy-preserving database queries or model parameters and inference data in privacy-preserving machine learning. All sensitive information should be safeguarded from being leaked to any unauthorized party (e.g., a cloud server).
  - *Integrity.* Integrity in secure computation means that no malicious party can sabotage the protocol execution and produce the wrong output for other parties. For example, in federated machine learning, a party may launch a backdoor attack to corrupt the performance of the training model by training its local model maliciously [156].
  - *Freshness.* Freshness in secure computation means that the involved party cannot leverage stale state or old data for stateful computation. For example, an adversary may launch a replay attack by re-transmitting stale valid messages to honest parties, thereby fooling the honest parties. Therefore, any messages transmitted between the parties should be passed with freshness checking, which ensures the state or the data is up to date.
  - *Fairness.* Fairness is a natural and desirable security requirement in SMC, which has shown its importance in auctions, contract signing, payment and other application scenarios. Fairness ensures that either all parties receive the protocol output or no one does [31]. While complete fairness for general computation is impossible [32], several possible and useful fairness solutions have been proposed such as gradual release [19], probabilistic fairness [91], optimistic exchange [11], fairness with penalties [3], and  $\Delta$ -fairness [112].
- **Adversarial models.** Adversarial models abstract attackers' power. In general, secure computation protocols consider two types of attackers: semi-honest and malicious. Specifically,
  - *Semi-honest.* A semi-honest adversary, also called honest-but-curious adversary, is assumed to follow the protocol's instructions honestly so that the tasks are conducted and completed correctly. However, the attacker is curious and tries to learn sensitive information from each step of the computation.
  - *Malicious.* A malicious adversary is stronger than the semi-honest one since it can arbitrarily deviate from the prescribed protocol. A malicious adversary could learn more information about the sensitive data through active attacks or save computation or storage costs by performing lazy computation and returning random results to other honest parties.

Besides, formal proof is an important aspect of evaluating security. A formal abstraction of the secure enclave was introduced in [112], which is also adopted by the subsequent works with formal security analysis, such as [31, 114, 150]. We present the formal TEE abstraction (also denoted  $\mathcal{G}_{att}$ ) in our supplementary material (Sec. B).



**2.2.4 Performance.** We present the performance assessment criteria from four aspects: experimental configuration, costs, accuracy, and open-sourceness, which are elaborated below. We should note that a fair comparison of secure computation protocols is not a trivial task, because the performance indicators may be obtained from different data sets and be exhibited in many different manners. Thus, we are not able to compare all the protocols under the same experimental configuration. Instead, we aim to provide a relatively objective comparison based on the following four aspects.

- **Experimental configuration.** Experimental configurations are diverse, such as computer specifications, platforms, workloads, data sets, and network architectures. Therefore, we will specify the necessary experimental configurations in the comparison, which would also help readers to understand the performances in real deployment.
- **Costs.** The secure computations exhibit their performances in many different ways, such as complexity estimation without experiments, absolute time consumption, a percent indicating relative cost, throughput, or others. To be as unified as possible, we present its baseline in the comparison. Specifically, the following baselines are considered.
  - *Plaintext model.* The plaintext model means that the function is computed outside the TEE without any protection, which exhibits exceptionally high performance. For comparing with the plaintext model, the protocols' performance will be worse, and we highlight the "slowdown" factor in comparisons.
  - *Fully in enclave.* The fully in enclave model means that the entire computational task is completed in the enclave, which may trigger the page swapping of the TEE and thus results in a large TCB size and worse performance. A well-designed protocol's performance will usually be better than the fully in enclave model, and we highlight the "speedup" factor in comparisons.
  - *State of the arts.* Some references compare their protocol with the state of the arts. In comparison, we highlight "slowdown" or "speedup" factors for degrading or improvement in performance.
- **Accuracy.** Accuracy is an essential criterion in many machine learning algorithms. Informally, accuracy is the fraction of predictions the model got right, and high accuracy is preferred. Accuracy in general secure computation means that the absolutely correct results should be returned. However, differential privacy based solutions adds noise perturbations to the results for meeting specific privacy requirements and thus may not produce absolutely correct results.
- **Open-Sourceness.** Last but not least, we also summarize whether an implemented protocol is open-sourced, which helps other researchers to understand the proposal better and to check its validity and performance.

## 2.3 Roadmap

Fig. 3 presents the roadmap of this survey. The completed survey consists of the primary part and additional supplementary material. In the primary part, TEE-based secure computation protocols for general-purpose function and machine learning are sequentially reviewed in Sec. 3 and Sec. 4, respectively. In the supplementary material, we review the TEE-based secure computation protocols for database queries in Sec. A. In each section, we present problem statements and research status with regard to different frameworks. For example, Sec. 3 reviews TEE-based protocols for general-purpose secure computation. Following the taxonomy presented in Sec. 2.1, we present the SOC, SDC, and SMC in Sec. 3.1, Sec. 3.2, and Sec. 3.3, respectively. After that, we provide thorough comparisons in Sec. 3.4 according to assessment criteria shown in Sec. 2.2. As two important and special research directions in secure computation, we organize contents for Machine Learning (in Sec. 4) and Database Queries (in Sec. A) in a similar way as in Sec. 3. Finally, concluding remarks and future directions are made in Sec. 5.

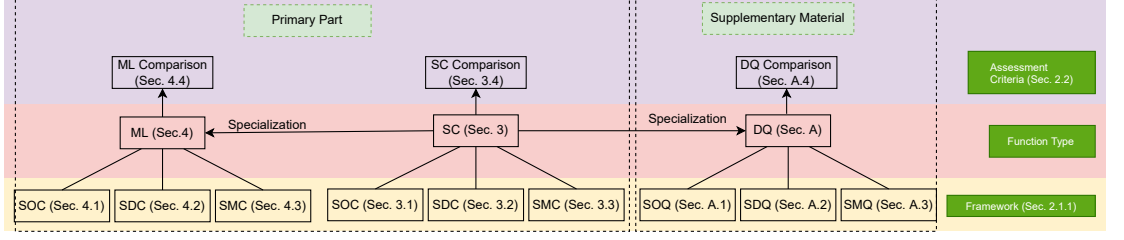


Fig. 3. Taxonomy, Assessment, and Organizations.

### 3 SECURE COMPUTATION USING TEES

In this section, we turn to discuss and compare existing general-purpose secure computation protocols using TEEs. According to different frameworks, we successively present secure outsourced computation, secure distributed computation, and secure multiparty computation.

#### 3.1 Secure Outsourced Computation

**3.1.1 Problem Statement.** Secure outsourced computation (SOC) allows a resource-limited client to outsource a computation task  $(\Phi, x)$  to a powerful cloud server, where  $\Phi$  is a function or a program for computation, and  $x$  is a sensitive input. To provide privacy guarantee, the task  $(\Phi, x)$  should be encrypted to  $(\llbracket \Phi \rrbracket, \llbracket x \rrbracket)$  and the cloud server finishes the computation in encrypted domain,  $(\llbracket y \rrbracket, \Gamma) = \text{SOC}(\llbracket \Phi \rrbracket, \llbracket x \rrbracket)$ , where  $\llbracket y \rrbracket$  is encrypted output and  $\Gamma$  is the proof for checking the integrity of execution. Upon receiving  $\llbracket y \rrbracket$ , the client decrypts it to the final output  $y$ . The *consistency* of SOC means that the final output  $y = \Phi(x)$ . The *privacy* means that the cloud server cannot learn any sensitive information about  $\Phi$  (function privacy),  $x$  (input privacy), or  $y$  (output privacy). The *integrity* means that the server correctly executes the task  $(\Phi, x)$ .

Shan et al. [128] surveyed the non-TEE approaches for SOC, such as fully homomorphic encryption, additive homomorphic encryption, random transformation, secret sharing, etc. Since the arising of trusted hardware, the TEE-based methods received a lot of attentions due to its high performance. As shown in Fig. 4, we provide the basic SOC workflow to explain the interactions between a client and a TEE-equipped server. The client builds a secure channel between the enclave by remote attestation, after which four steps are followed. (1) The client loads the program  $\Phi$  into the enclave, and (2) The enclave responds with a triple  $(b, \Phi, \Gamma_1)$ , in which  $\Gamma_1$  proves the program is loaded successfully ( $b = 1$ ); otherwise,  $b = 0$  and  $\Gamma_1 = \perp$ . (3) The client submits an input  $x$  to the enclave. (4) The enclave returns the final results  $(y, \Gamma_2)$ , in which  $y = \Phi(x)$  and  $\Gamma_2$  proves the program is executed honestly. Note that all messages between client and enclave are sent to each other over the secure channel.

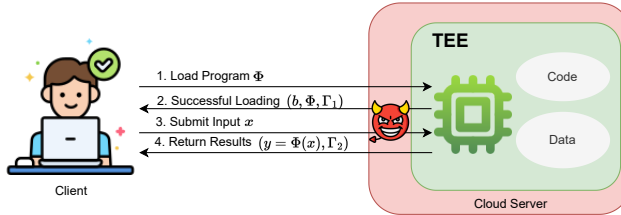


Fig. 4. Secure Outsourced Computation

**3.1.2 Research Status.** In [12], Barbosa et al. presented the notion *attested computation*, which focuses on integrity assurance by building a secure channel via key exchange. With their proposed *utility theorem*, they presented the first SOC framework using trusted hardware, which also offers formal security analysis by simulation-based proof. Another formal treatment for TEE-based SOC is the symbolic model [70, 137], which aims to design secure remote attestation execution protocols. To against side-channel attacks, ObliCheck [135] employs symbolic execution to check whether all execution paths exhibit the same observable behavior. In [112], Pass et al. presented the *formal abstractions and rigorous exploration* of TEE and offered a TEE-based stateful obfuscation to show the power of attested execution. Then, for the first time, they described the basic  $\mathcal{G}_{att}$  abstraction (Sec. B in supplementary material) capturing the essence of TEE. After that, Pass et al. employed the abstraction to design an SOC protocol, which was proved to be UC-secure in  $\mathcal{G}_{att}$ -hybrid model. Following [112], many works in hardware-supported secure computation protocols were formally treated and proved to be UC-secure by employing  $\mathcal{G}_{att}$  abstraction (e.g., [150]).

To mitigate rewind attack, a public ledger (PL) served as a trusted data source is utilized to ensure the TEE state freshness while computing a stateful function [72]. The authors proposed a hybrid computing framework in which each input is registered from the public ledger. The public ledger signs the inputs by a signature scheme, and then the enclave checks the freshness of the state according to the public ledger before executing the general function computation. Therefore, it achieves stateful, interactive functionality, even on devices without persistent storage (such as mobile devices).

The fairness in SOC is different from the one in SMC. In [40], Dang et al. explored the fair exchange in a marketplace for secure outsourced computation. In the market, the client first commits a remuneration and then submits a computational task to computing nodes. The computing nodes get the remuneration if the task is completed successfully. To achieve the goals, the authors proposed a framework Kosto, which introduced a third party (called a broker) to assist the clients in attesting correct instantiation of the enclaves and computing nodes' certain commission fees.

## 3.2 Secure Distributed Computation

**3.2.1 Problem Statement.** Secure distributed computation (SOC) is run between a master node and multiple slave nodes. Generally, each slave node computes an intermediate result over its inputs, and then the master node aggregates all intermediates into the final results. Specifically, the task  $(\Phi, x)$  is split into many sub-computational tasks  $(\Psi, x_i)$ ,  $i = \{1, 2, \dots, N\}$  and an aggregation task  $\Pi$  such that

$$\Phi(x) = \Pi(\Psi(x_1), \Psi(x_2), \dots, \Psi(x_N)) \quad (1)$$

Each slave node runs a sub-computational task, and the master node takes charge of the aggregation task. According to different secure computation protocols, the algorithms  $\Psi$  and  $\Pi$  may work interactively. Note that privacy and integrity also should be achieved in an SDC protocol.

Shyuan et al. [101] surveyed the non-TEE approaches for SDC, such as linear secret sharing, homomorphic encryption, and random transformation, etc. These solutions may be used on different platforms such as MapReduce and Spark. Unlike the above approaches, we survey the hardware-supported technologies for SDC protocols. As shown in Fig. 5, we provide the basic SDC workflow to explain the interactions between the master node and multiple slave nodes. (1) The client builds secure channel with servers (including master and slave nodes) via remote attestation, and then ensures distributed program  $\Psi$  and aggregation program  $\Pi$  are successfully loaded into the slave nodes' enclave and the master node's enclave, respectively. (2) The client submits an input  $x$  to the master node. (3) The master node splits the inputs  $x$  into  $x_1, x_2, \dots, x_N$  and then distributes  $x_i$  to  $i$ -th slaves. (4) Each slave node finishes the sub-computational tasks in TEE and responds

to the master node with a tuple  $(\Psi(x_i), \Gamma_i)$ , in which  $\Psi(x_i)$  is the  $i$ -th intermediate result and  $\Gamma_i$  proves slave node executes the program  $\Psi$  honestly. (5) Finally, the master node aggregates all intermediate results by  $\Pi$  and then sends the final results  $(y, \Gamma)$  to the client, in which  $\Gamma$  proves the program  $\Pi$  is honestly executed. Note that all messages among the entities are exchanged over the secure channel.

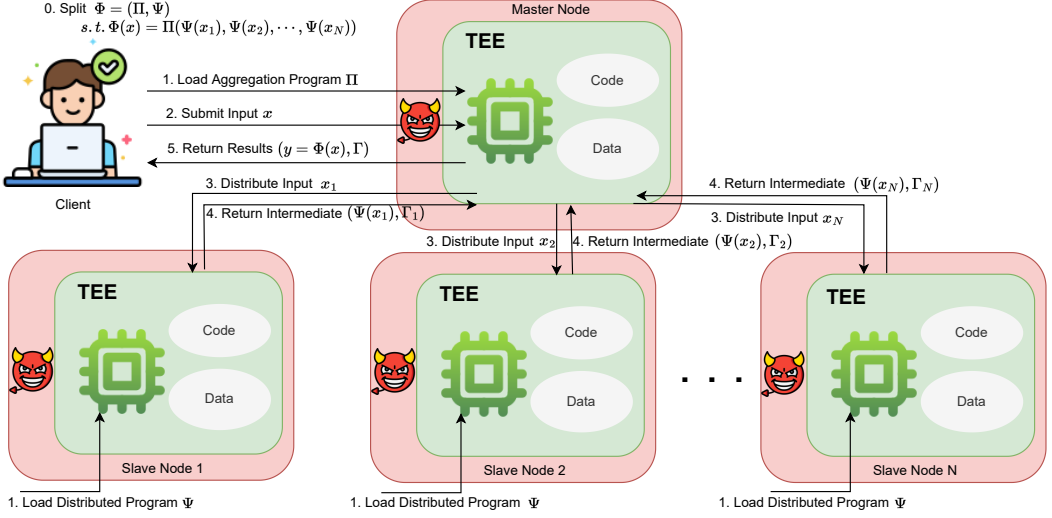


Fig. 5. Secure Distributed Computation

**3.2.2 Research Status.** In [126], a TEE-based framework VC3 for secure distributed computation in the cloud was proposed by Schuster et al. VC3 enables users to finish computations in the distributed MapReduce setting while maintaining the confidentiality of data and ensuring the accuracy and completeness of the final results. VC3 does not modify the original Hadoop and thus maintains compatibility with Hadoop. In the design of VC3, the trusted computing base (TCB) excludes the operating system and the hypervisor; therefore, it achieves confidentiality and integrity even if the host is corrupted. Relying on the well-designed safe r/w operators inside the enclave to unsafe memory, VC3 facilitates the deployment of new SDC protocols under the MapReduce platform. However, it does not solve the access pattern leakage and does not support multi-user key exchange, which may result in huge overhead in the remote attestation. On top of VC3, Ohrimenko et al. in [105] observed that intermediate traffic might leak sensitive information by employing geographical data as the motivating example. They proposed a mechanism to hide the correlation between the map function and the reduce function against a Map-Reduce game and then prove security by a heuristic method.

To overcome the limited memory of TEEs, an ORAM controller is installed inside the TEE and an ORAM storage is arranged in external memory or disk. By contrast, M2R proposed in [45] focuses on resisting the side-channel attacks in SDC where each unit is equipped with trusted hardware. M2R allows adversaries to obtain channel information and launch passive attacks (dataflow patterns, order of execution, time of access) and active attacks (tuple tampering, misrouting tuples). While expensive ORAM introduces a multiplication factor  $O(\log N)$  overhead to the latency, M2R incurs only an additive logarithmic factor by grouping then shuffling technique, which employs the

shuffler in the MapReduce platform directly without degrading privacy. However, all the above schemes only work on small-scale data (e.g., 1-5GB).

To build a UC-secure SDC protocol, OblIDC [151] is an SGX-based framework that introduces the ODC-privacy to capture the privacy leakage in the entire life circle, which is modeled as a data-flow graph. Specifically, it decomposes the distributed computing into four phases: job deployment, job initialization, job execution, and results return, and for each step, a UC-secure two-party protocol was designed to meet ODC privacy. It is worth mentioning that oblIDC achieves both semantic security and oblivious traffic even if the adversaries are malicious. Furthermore, OblIDC supports any general-purpose computation by defining the sensitive C-style code to be run in an enclave.

### 3.3 Secure Multiparty Computation

**3.3.1 Problem Statement.** Secure multiparty computation (SMC) is aimed at computing a joint task among a number of distinct yet connected parties while revealing nothing but the output. Specifically, assume each participant has an input  $x$ . In SMC protocol for a task  $\Phi$ , the correctness means that  $\Phi(x_1, x_2, \dots, x_n)$  is computed correctly, and the privacy means that the execution of the protocol only reveals the final result to each party. Integrity ensures that all parties follow the protocol steps honestly. Some applications may enforce the SMC to be fair, which means that malicious parties receive the outputs if and only if other honest parties also receive their outputs. SMC has shown its importance in many application domains, such as privacy-preserving bidding, private DNA comparison, privacy-preserving machine learning, etc.

Choi et al. [29] surveyed the non-TEE approaches for SMC. Since Yao's Garbled Circuits [155], many researchers have targeted designing a secure and efficient SMC protocol. Nevertheless, the solution is still impractical because of the huge communication or extensive computational cost. Many existing non-hardware solutions focus on improving efficiency, such as Sharemind [20], VMCrypt [93], ABY [42], SPDZ [38], HyCC [24], etc. Most of them rely on the primitives such as GC-OT, GMW, FHE, Secret Sharing, ORAM, etc. In this survey, we review the approaches relying on TEE.

Secure two-party computation workflow is presented in Fig. 6, which can be easily extended to multiple parties. Both parties equip TEEs and build a secure channel via remote attestation. Taking their own data  $x_1$  and  $x_2$  as inputs, two parties finish the computation in an interactive manner, which consists of multiple rounds of communication. At each round, a message associated with a proof  $(m_i, \Gamma_i)$  is sent to another party where  $m_i$  is a committed message to ensure the privacy of the execution, and  $\Gamma_i$  is employed to prove that  $m_i$  is computed honestly. Finally, the two parties may run a fair protocol to distribute the final result  $\Phi(x_1, x_2)$ . Note that all messages between the two parties are sent over the secure channel.

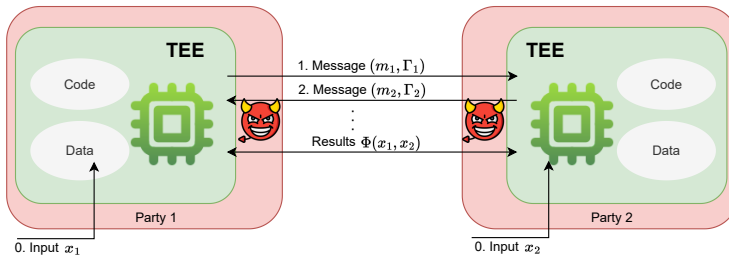


Fig. 6. Secure Two Party Computation

**3.3.2 Research Status.** In [60], Gupta et al. showed the naive execution of functions within TEE might result in significant side-channel leakages, such as runtime and RAM access patterns. Instead of relying on SGX enclaves to process the entire computation, the authors leveraged the garbled circuits to handle the sensitive portion of the computation and advocate separating the computation into separate circuits and SGX components. In such a way, the secrets are protected even if the enclave is compromised. However, they did not treat the SMC problem formally. Bahmani et al. in [7] extended the attested computation [12] to labeled attested computation (LAC), which makes each party a label. Thus users can get attestations of parts of code corresponding to specific labels. With the proposed LAC, they built an efficient SMC protocol from SGX by composing attested key exchange procedures for each participant in parallel. Furthermore, the TEE-based SMC protocol was proved by simulation, and some benchmarks compared to ABY [42] are presented to show its efficiency.

Based on the  $\mathcal{G}_{att}$ -hybrid abstraction [112], Pass et al. drew several surprising results on SMC. (1) UC-secure SMC protocol is impossible if at least one party is not equipped with a TEE. However, global Augmented Common Reference String (ACRS) [25] model makes the UC-secure SMC realizable even if only one party is equipped with a TEE because it allows simulating corrupted clients during proof. (2) Combined with secure key exchange, UC-secure 2-party computation is possible when both parties have TEEs. (3) Combined with secure key exchange, if both parties have TEEs and trusted clocks, then fair 2-party computation is achievable. (4) Fairness is impossible for general functionalities when one party is not clock-aware, but fairness for certain functions would be possible with the help of the ACRS model.

To alleviate the burden of trust on the enclave, Choi et al. [30] explored a balanced method to split a function  $f$ . Instead of evaluating the entire function  $f$  in the enclave, the proposed 2P-SFE protocol allows the designer to choose which components should be evaluated within the enclave. 2P-SFE modeled the enclave as a black box and was applied to two practical problems: private queries to a database and private navigation based on Dijkstra's algorithm. However, the scalability of 2P-SFE is limited. While 2P-SFE only works for boolean circuits, the authors in [51] proposed a method to evaluate the universal circuit via Intel SGX, where the universal circuit can emulate any function by programming input bits to a given size. However, both [30] and [51] assume the parties are semi-honest. In [150], Wu et al. proposed a generic framework HYBRTC, which introduces the concept of TEE trust levels. Specifically, in an SMC protocol, a party might fully trust TEE, while another party may only trust the TEE partially or never trust it. HYBRTC solves the problem when different levels of trust exist in an SMC protocol. From a high-level overview, for parties who think TEEs are reliable, their private inputs can be processed in the enclave. Otherwise, their private inputs should be protected by a cryptographic SMC protocol. Besides, HYBRTC also eliminates side-channel attacks.

**Fairness in SMC.** TEE was also explored to design fair SMC protocols. In [31], Choudhuri et al. presented a hybrid model for achieving *fairness* by using a public ledger, even in the case of a dishonest majority. Note that existing solutions such as blockchains or Google's certificate transparency logs can implement the public ledger. What important is that the proposed SMC has complete fairness, which is different from the method of penalizing an aborting party or achieving weaker notions such as  $\Delta$ -fairness. After that, [114] presented a robust construction that ensures fairness, although it allows to output an incorrect value. While implementing fairness by leveraging a public ledger, the input, output and the state need to be stored on the chain, which brings the burden of the blockchain. In [132], LucidiTEE explored a method that does not store inputs, outputs or states on the chain. Besides, LucidiTEE also suggested integrating the history-based policy into SMC, which achieves fair delivery and enforces policy-restricted input.

Table 1. Comparisons of Secure Computation

Schemes	Settings		Methodology		Security						Performance	
					Security Goals				Adversarial Models	Formal proof		
	Framework	Function	Technologies	Primitives	Privacy	Integrity	Fairness	Freshness			Configurations	Costs
[M2R [45], USENIX Security, 2015]	SDC, n	Key-value Pair Analysis	Stateless TEE + OP	PRP, Sig, PKE, SecMix	IO, pattern, IR	IO, exe, Completeness	✗	✗	Malicious Servers HBC client	✗	Baseline: Fully in Enclave Data size: 2-4GB	0.17-1.0x slowdown
[VC3 [126], SP, 2015]	SDC, n	General	Stateless TEE	PRF, Sig, PKE, KE	IO, IR	IO, exe, Completeness	✗	✓	Malicious Servers HBC client	✗	Baseline: Plaintext Model Datasize: 1GB	WordCount: 0.05-0.4x slowdown
[OCF [105], CCS, 2015]	SDC, n	Key-value Pair Analysis	Stateless TEE + OP	PRF, Sig, PKE, KE, PRP	IO, pattern, IR	IO, exe, Completeness	✗	✗	Malicious Adaptive Servers HBC client	✗	Baseline: Plaintext Model Datasize: 2GB	Aggregate: 2x slowdown
[2P-SFE [60], FC, 2016]	SDC, 2	Stateless Function	Stateless TEE	GC	IO	exe	✗	✗	Malicious Server	✗		✗
[Att. Comp. [12], EuroS&P, 2016]	SOC	General	Stateful TEE	KE, AE	IO, state	IO, exe	✗	✗	Malicious Server	SIM		✗
[Labeled Att.Comp. [7], FC, 2017]	SMC, n	General	Stateful TEE	KE, AE	IO, state	IO, exe	✗	✓	Malicious Parties	SIM	Baseline: ABY	HD: 5-10x speedup PSI: 8-200x speedup
[PST17-I [112], EUROCRTP, 2017]	SOC	General	Stateful TEE	Sig, AE	IO	IO, exe	✗	✓	Malicious Server	UC		✗
[PST17-II [112], EUROCRTP, 2017]	SMC,2	General	Stateful TEE	Sig, PKE, AE, NIZK	IO	IO, exe	✗	✓	Malicious Parties	UC		✗
[PST17-III [112], EUROCRTP, 2017]	SMC,2	General	Stateful TEE	Sig, PKE, AE, NIZ, Clock	IO	IO, exe	Δ-Fairness	✓	Malicious Parties	UC		✗
[FSMC [31], CCS, 2017]	SMC, n	General	Stateful TEE + PL	OWF, Sig, AE, Com	IO	IO, exe	✓	✗	Malicious Parties	UC		✗
[2P-SFE <sup>†</sup> [68], AsiaCCS, 2018]	SMC, 2	Garbled Circuits	Stateful TEE	GC, OT	IO, IR	IO, exe	✗	✗	HBC parties	SIM	Baseline : GC	Dijkstra: 1.01-200x speedup
[KGM [72], NDSS, 2019]	SOC	General	Stateless TEE + PL	PRF, DAE, Com	IO, state	IO, exe, state	✗	✓	Malicious Server	SIM		✗
[Kosto [40], ESORICS, 2019]	SOC	General	Stateful TEE	PRF, AE, Com	IO	IO, exe	✓	✗	Malicious Parties	✗	Baseline: Plaintext Model	mcf: 2x slowdown deepsjeng: 2.3x slowdown leela : 2.1x slowdown xz : 1.2x slowdown exchange: 0.4x slowdown
[ObliDC [151], AsiaCCS, 2019]	SDC, n	General	Stateful TEE + OP	Sig, AE, PKE	IO, pattern, IR	IO, exe, Completeness	✗	✓	Malicious Adaptive Servers HBC client	UC	Baseline: Plaintext Model Datasize: 2GB	WordCount: 0.8-1.2x slowdown RandomWriter: 0.3-1.42x slowdown
[LucidTEE [132], Preprint, 2019]	SMC, n	General	Stateless TEE + PL	PKE, Sig, AE, Com	IO, state	IO, exe, state, policy	✓	✓	Malicious Parties	UC	No baseline Datasize: 30MB	PSI: 10s
[WNS22 <sup>‡</sup> [150], NDSS, 2022]	SMC, n	SELECT-JOIN	Stateful TEE	PRF, HE, BF, CHF	IO, pattern	IO, exe	✗	✓	Malicious Parties	UC	Baseline: ABY Function: Select-join	Select-join: 2-18kx speedup

GC: Garbled Circuits, KE: Key Exchange, SDC,*n*: Secure Distributed Computation with *n* Slave Nodes, SOC: Secure Outsourced Computation, SMC,*n*: Secure Multiparty Computation with *n* Parties, AE: Authenticated Encryption, SIM: Simulate-based Proof, HD: Hamming Distance, PSI: Private Set Intersection, Sig: Signature Scheme, PKE: Public key encryption, NIZK: Non-interactive Zero-knowledge Proof, OWF: One Way Function, Com: Commitment Scheme, PRF: Pseudo-random Functions, DAE: Deterministic Authenticated Acription, HE: Homomorphic Encryption, BF: Bloom Filter, CHF: Cuckoo Filter, IR: Intermediate Results, PRP: Pseudo-Random Permutation, SecMix: Secure Mixer, OT: Oblivious Transfer;

<sup>†</sup> <https://github.com/FICS/smcsgrx>,

<sup>‡</sup> <https://github.com/HybrTC/HybrTC.git>;

### 3.4 Comparisons

A comprehensive comparison is made in Table 1 to analyze existing secure computation protocols with regard to the setting, methodology, security features, and performance. Since TEE provides basic confidentiality and integrity guarantees, most of the existing secure computation protocols are against malicious parties. Only LucidiTEE [132] achieves fairness and freshness simultaneously; however, it only works on a very small-scale data set (around 30MB). Due to the different settings and experimental configurations, we are unable to give an explicit performance comparison among the referenced works in Table 1. Instead, we present the performance comparison between a protocol and its baseline protocol. For example, WNS22 [150] improves the performance by about 2-18k times compared to its baseline ABY [42] for the select-join function.

## 4 SECURE MACHINE LEARNING USING TEES

This section turns to review and compare TEE-based protocols towards secure machine learning. Following the frameworks presented in Sec. 2, the protocols are classified into secure outsourced learning, secure distributed learning, and secure multiparty learning. In secure outsourced learning, we consider two types of computational tasks: training and inference. Training refers to using training data sets and a machine learning algorithm to generate a trained machine learning model. Inference refers to using a trained machine learning model to make a prediction for a given data item. Secure training and inference are also called privacy-preserving Machine Learning as a Service (MLaaS) [64]. We found that approaches in secure training can also apply to secure inference, therefore we do not consider secure inference in distributed and multiparty learning. Instead, we only consider secure inference under the outsourced learning setting.

### 4.1 Secure Outsourced Learning

**4.1.1 Problem Statement.** We review secure outsourced learning from two aspects: secure outsourced training (SOT) and secure outsourced inference (SOI). SOT enables a resource-limited client to outsource a training task  $(\Phi, \llbracket X \rrbracket)$  to a single cloud server with powerful resources and generate a trained model  $M$ . Note that  $X = \{(x_i, y_i)\}_{i=1}^N$  is an ensemble of data item (e.g.,  $x_i$ ) with its label (e.g.,  $y_i$ ). To protect the privacy of training data and model, the cloud server performs training operations  $\Phi$  on encrypted data sets  $\llbracket X \rrbracket$  and outputs  $(\llbracket M \rrbracket, \Gamma)$ , where  $\llbracket M \rrbracket$  is an encrypted trained model, and  $\Gamma$  is a proof of integrity assurance. The *correctness* of SOT means that the final result  $M = \Phi(X)$ . In other words, the result of SOT is equivalent to the one trained locally. *Privacy* means that the cloud server cannot learn anything sensitive information about  $X$  (input privacy) or  $M$  (model privacy, known as output privacy in SOC). The *integrity* enforces the cloud server performing  $\Phi(\llbracket X \rrbracket)$  honestly by generating a proof  $\Gamma$  to prove the integrity of execution.

While cryptographic primitives (e.g., homomorphic encryption, secret sharing, secure multi-party computation) can support secure training [21], they introduce high computation and communication overhead [48]. TEE-based SOT has gain its popularity in recent years due to its appealing performance advantages. As shown in Fig. 7a, we provide a general workflow of SOT to explain the interactions between a client and a TEE-equipped cloud server. The client builds a secure channel between the enclave by remote attestation, after which four steps are followed. (1) The client loads the program  $\Phi$  to the enclave via a secure channel by remote attestation. (2) The enclave responds with a triple  $(b, \Phi, \Gamma_0)$ , in which  $\Gamma_0$  proves the program is loaded correctly ( $b = 1$ ); otherwise,  $b = 0$  and  $\Gamma_0 = \perp$ . (3) The client submits an encrypted data ensemble  $\llbracket X \rrbracket$  to the enclave. (4) The enclave cooperates with the untrusted cloud server to generate the final results  $(\llbracket M \rrbracket, \Gamma)$ , where  $M = \Phi(X)$  and  $\Gamma$  proves the program is executed honestly. The enclave returns the final results to the client.



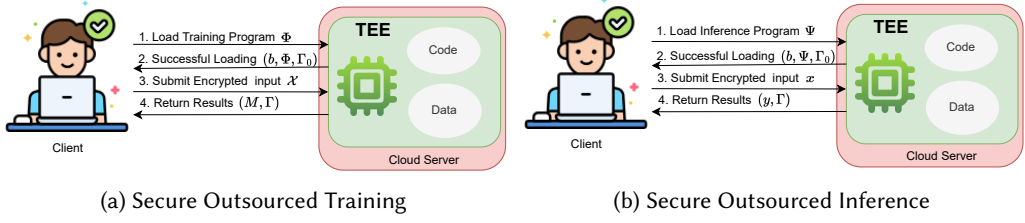


Fig. 7. Secure Outsourced Learning

In SOI, a single cloud server holds the trained model  $M$  and allows a resource-limited client to submit an inference task  $(\Psi, \llbracket x \rrbracket)$ , and then responds the client with an inference output  $y$ . Note that here  $x$  is just a single data item and the output  $y$  is the prediction result. To protect the privacy of  $x$ , the cloud server performs inference operations  $\Psi$  on encrypted inference data  $\llbracket x \rrbracket$ . It outputs  $(\llbracket y \rrbracket, \Gamma)$ , where  $\llbracket y \rrbracket$  is an encrypted inference result, and  $\Gamma$  is a proof of integrity for operations. The *correctness* of SOI means that the final inference result  $y = \Psi(x, M)$ . In other words, the result of SOI is equal to the one that finishes the inference task locally. The *SOI privacy* means that the cloud server cannot learn anything sensitive information about  $x$  (input privacy) or  $y$  (output privacy). In case the cloud server has no ownership of the model  $M$ , the encrypted model  $\llbracket M \rrbracket$  is fed to server to achieve the model privacy. The *integrity* enforces the cloud server to perform  $\Psi(\llbracket x \rrbracket, M)$  honestly and generate the integrity proof  $\Gamma$ .

Fig. 7b shows a general workflow of SOI and the interactions between a client and a single cloud server that equips TEE and holds a machine learning model  $M$ . The client builds a secure channel between the enclave by remote attestation, after which five steps are followed. (1) The client loads the program  $\Psi$  to the enclave via a secure channel. (2) The enclave responds with a tuple  $(b, \Psi, \Gamma_0)$ , in which  $\Gamma_0$  proves the program is loaded successfully ( $b = 1$ ); otherwise,  $b = 0$  and  $\Gamma_0 = \perp$ . (3) The client submits an encrypted input  $\llbracket x \rrbracket$  to the enclave. (4) The enclave cooperates with the cloud server to generate the final results  $(\llbracket y \rrbracket, \Gamma)$ , in which  $\llbracket y \rrbracket = \Psi(\llbracket x \rrbracket, M)$  and  $\Gamma$  proves that the program is executed honestly. The enclave returns the final results to the client.

**4.1.2 Research status of SOT.** In [106], Ohrimenko et al. combined SGX and data-oblivious primitives to design data-oblivious machine learning training algorithms, such as support vector machines, matrix factorization, neural networks, decision trees, and k-means clustering. Experimental evaluations show that its performance outperforms the MPC-based solutions. However, the approach results in a large TCB, which limits its use to support large machine learning models. To reduce TCB and enable the TensorFlow framework, Kunkel et al. [81] presented the TENSORSCONE framework to support secure machine learning computations on untrusted infrastructure. TENSORSCONE can achieve more than 80% precision compared with GPU-only baseline training. In [157], Peterson et al. proposed Plinius that combined the features between persistent memory (PM) and Intel SGX enclave. Plinius achieves 3.2-3.7 $\times$  performance improvement for saving and restoring models on real PM hardware, respectively. Besides, GPU-based TEEs [146] were introduced to accelerate secure training (e.g., Telekine [67]). To enable parallel GPU acceleration, the proposed AsymML in [103] partitions a target machine learning model into trusted and untrusted parts. Thus, AsymML can provide more than 5 $\times$  speedup than pure TEE-based training. Besides, model partition is also a popular technology to accelerate the TEE-based training [5, 63, 157].

To enable realistic integrity-preserving deep neural network (DNN) model training for heavy workloads, Asvadishirehjini et al. [5] presented GINN that combined random verification of selected computation steps with systematic adjustments of DNN hyperparameters to limit the attacker's

ability to shift the model parameters arbitrarily. GINN can achieve high integrity and 2-20 $\times$  performance improvement over a pure TEE-based training method. In [63], to optimize the performance, Hashemi et al. presented DarKnight framework that uses TEE to provide privacy and integrity verification and GPUs to perform the bulk of linear algebraic computation. DarKnight [63] proposed a customized data encoding strategy based on matrix masking, which is adopted to create input obfuscation within the TEE. DarKnight achieves an average of 6.5 $\times$  training speedup compared with the pure TEE-based solutions. Goten [102] adopted a dynamic quantization scheme to cater for the fluctuation in weight during training to optimize performance. It shows a 6.84-132.64 $\times$  speedup than a pure TEE-based solution and the latest secure multi-server cryptographic solution [147]. In [107], Ozga et al. presented PERUN that considers multi-stakeholder collaboratively generating a trained model where the stakeholders consist of the training data owner, training code owner, model owner, and inference code owner. Each entity outsources the computation to a cloud server and provides a service to other entities. PERUN gives a 161-1,560 $\times$  performance improvement compared with a pure TEE-based method.

**4.1.3 Research status of SOI.** Due to the similarity of computation between training and inference, many existing works [63, 67, 102, 103, 107, 157] support SOT and SOI simultaneously. In the following, we review other works focusing on SOI. In [58], Grover et al. first pointed out the inference of DNN based on Intel SGX enclaves faced with access pattern based attacks, so this work designed input-oblivious PRIVADO framework to address the problem and support secure and integrated inference services. PRIVADO with Torch framework incurs an average 17.18% overhead on 11 different DNN models. The work [59] formulated the information exposure problem as a reconstruction privacy attack for a cloud-based inference service and quantified the adversary's capabilities with different attack strategies. DeepEnclave was proposed in [59] to partition a deep learning model into a FrontNet and a BackNet, where the FrontNet is enforced to do enclaved execution, and the BackNet runs out of secure enclaves. DeepEnclave causes 1.64-2.54 $\times$  overhead due to the enclaved execution of the FrontNet compared to GPU execution of the FrontNet. To optimize the performance of inference, Florian et al. proposed SLALOM [142] that partitioned DNN computations into trusted and untrusted parts. All the computation of linear layers of a DNN model are executed in an enclave, while other computation of the DNN model are loaded to an untrusted GPU for acceleration. Although SLALOM adopts some cryptographic primitives to support verifiable and private inference, it achieves 4-20 $\times$  performance improvement compared with a pure TEE approach.

To break through the memory limitation of the enclave and accelerate an inference, Origami [100] combines enclave, cryptographic tools, and GPU/CPU acceleration. The enclave adds noise to obfuscate inference data and sends the obfuscated data to an untrusted GPU/CPU for computation acceleration. Origami achieves 11 $\times$  performance improvement than a pure SGX approach, and 15.1 $\times$  performance improvement compared to SLALOM [142]. Aiming to address the memory limitation and page swapping of SGX enclave, Lee et al. [84] carefully developed on-demand weights loading, memory-efficient inference, and parallel processing pipelines in their proposed OCCUMENCY, which gives a 3.6 $\times$  speedup compared to a pure TEE-based method and 72% latency overhead compared to a pure GPU approach. In [125], Alexander et al. presented the ENNCLAVE tool-chain to cut TensorFlow models at any layers and split them into public and enclave layers, where GPU performs public layers for acceleration. In contrast, the enclave executes private layers for security and integrity. Accuracy results of ENNCLAVE are close to a pure GPU method. Aiming to solve the large memory allocation and low memory re-usability problem in deep learning systems, [74] presented VESSELS that optimized the memory management of SGX. However, in contrast to a pure GPU-based inference method, VESSELS results in about 3-20 $\times$  inference latency.

To provide dependable and timely inference services, Xiang et al. [152] proposed AegisDNN that adopted a dynamic-programming algorithm based on the layer-wise DNN time and Silent Data Corruption profiling mechanism to find a layer protection configuration for each inference task. AegisDNN supports Caffe, PyTorch, and TensorFlow frameworks and achieves up to 88.8% less failure rates than the pure GPU-based approach and 99.9% shorter relative response time than the pure enclave-based approach. Different from the previous works, [111] focused on the fairness of inference for the machine learning model with the enclave.

## 4.2 Secure Distributed Learning

**4.2.1 Problem Statement.** Secure distributed training (SDT) enables  $n$  ( $n \geq 2$ ) slave nodes to jointly perform a training task  $(\Phi, X_1, \dots, X_n)$  with the assistance of a master node and generate a trained model  $M$ , where  $X_i$  is the  $i$ -th party's training data. Particularly,  $\Phi$  consists of slaves' local training operation  $\Phi_i$  and master's aggregation operation  $\Pi$  such that

$$\Phi(x) = \Pi(\Phi_1(X_1), \Phi_2(X_2), \dots, \Phi_N(X_N)). \quad (2)$$

Note that each slave node performs a local training  $\Phi_i$  and outputs intermediate result  $(\llbracket m_i \rrbracket, \Gamma_i)$ , where  $\llbracket m_i \rrbracket$  is an encrypted intermediate model (also called local model), and  $\Gamma_i$  proves the local training is executed honestly. After that, master node aggregates the local models into one global model. The *privacy* means that master node and slave nodes fail to learn any sensitive information about  $X$  (input privacy) or  $M$  (output privacy). The *integrity* enforces the master node and the slave nodes to honestly perform  $\Pi$  and  $\Phi_i$  and generate the respective proofs.

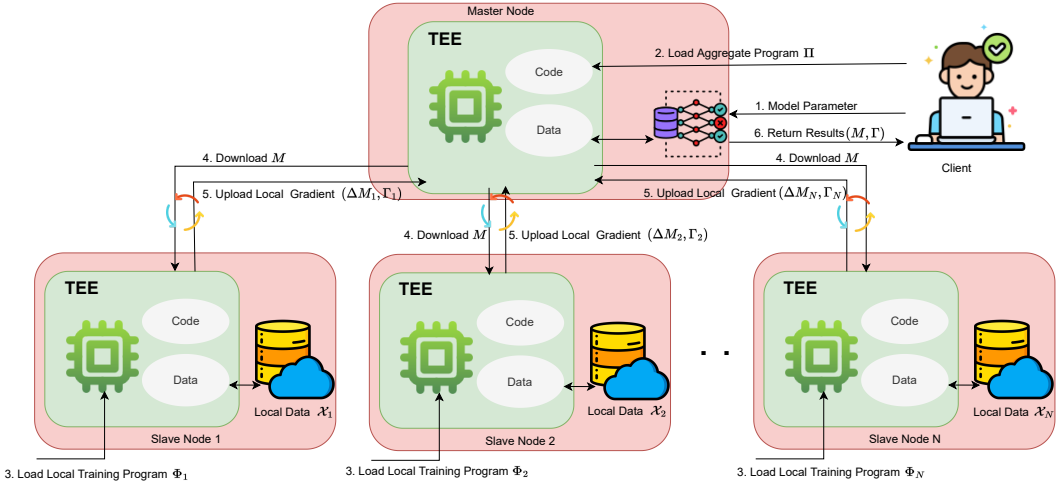


Fig. 8. Secure Distributed Training

As depicted in Fig. 8, we present a general workflow of TEE-based SDT to explain the interactions among a client, multiple TEE-equipped parties, and a TEE-equipped master node. The client builds a secure channel between the master node's enclave via remote attestation, after which six steps are followed. (1) The client uploads the model parameter to the master node. (2) The client ensures the aggregation program  $\Pi$  is loaded to the master node's enclave successfully and (3) ensures the local training algorithm  $\Phi_i$  is loaded into slaves' enclave successfully. (4) The slave nodes download the initial global model  $M$  and then generate a model gradient  $\Delta M$ . (5) The slave nodes submit the gradient  $\Delta M$  to the master node. Note that the steps (4) and (5) are repeated until a convergent

model is generated or the maximum number of iterations is reached. (6) Finally the master node returns the final global model to the client.

Note that Federated Learning (FL) also belongs to the distributed framework. FL serving as a special scenes of SDT has recently attracted much attention from the research and industrial community. The input data in SDT can either be encrypted or unencrypted. However in FL, each slave has its own training data set and a local model over local unencrypted data. The *privacy* for FL means that master node fails to learn any sensitive information about  $X$  (input privacy) or  $M$  (output privacy). The *integrity* enforces the master node and slave nodes performing  $\Pi$  and  $\Phi_i$  honestly and generating the proof accordingly, respectively.

**4.2.2 Research Status.** In [68], the proposed Chiron adopted SGX and the sandbox Ryoan [69] to perform model training and protect the privacy of both training data and the trained model, where Ryoan sandbox was used to prevent leaking the training data outside the enclave. Particularly, Chiron supports distributed training with multiple enclaves. Compared with training deep neural networks on CPU-only, Chiron increases about 4%-20% training cost. The work [158] proposed Citadel that allows parties (or say data owners) holding training data and an aggregation server (or say aggregator owner) equipped with enclaves to protect their input privacy. When multiple enclaves are deployed, Citadel achieves a less than  $1.73\times$  performance slowdown. In contrast to other solutions, [66] proposed StarFL that combines multiple techniques, such as TEE, MPC, and satellites (who are responsible for distributing keys). When the last four layers are inside enclave execution, the running time of StarFL is no more than 5% higher than a pure CPU-based training approach. Brito et al. presented SOTERIA [23], a distributed privacy-preserving machine learning (ML) system based on TEEs. SOTERIA made performance trade-offs for the distributed Apache Spark framework and its ML library. As SOTERIA supports executing non-sensitive operations outside an enclave, it improves the runtime performance of distinct ML algorithms by up to  $1.7\times$  compared to a pure SGX-spark method.

To support unmodified TensorFlow applications, Quoc et al. presented a distributed secure machine learning framework (SECURETF) based on TensorFlow for the untrusted cloud infrastructure [119]. In contrast to Chiron [68], SECURETF support SDT and SDI simultaneously and enjoys the advantage of unmodified TensorFlow applications. But it endures roughly  $14\times$  training overhead compared to a pure GPU approach.

**Federated Learning.** SecureFL [82] equips slave nodes and the master node of FL with TEEs. However, due to the partial in-enclave training, SecureFL results in 1.6%-23.6% overhead compared with a pure CPU-based training. Aiming to limit privacy leakages in FL, Mo et al. presented a privacy-preserving federated learning (PPFL) framework [98] for mobile systems using TEE. PPFL adopted TEE to train each layer of the model until its convergence by leveraging the greedy layer-wise. Thanks to the TEE, PPFL can defend against data reconstruction, property inference, and membership inference attacks. In contrast to the standard FL without privacy protection, PPFL introduces about  $1.3s-1.5\times$  training overhead.

To improve the efficiency, TrustFL proposed in [160] suggested using GPU to train a local model for each party and then verifying the correctness in an enclave. TrustFL utilized a "commit-and-prove" MHT-based commitment scheme to verify the integrity of the training algorithm and a "dynamic-yet-deterministic" data selection strategy for fresh training data. Thanks to GPU-based training, TrustFL provides 1-2 orders of magnitude speedup than a pure TEE-based FL method. Aiming to defend against Byzantine failures in FL and enable secure model aggregation, Zhao et al. proposed SEAR based on TEE [163]. The key idea of SEAR is to build a secure channel between a party and the aggregation server by remote attestation, and encrypted local models are aggregated in the aggregation server's enclave. SEAR achieves a  $4\times-6\times$  performance improvement compared

to the existing secure aggregation framework. ShuffleFL [161] is proposed to protect the privacy of model gradients and defend against side-channel attacks. It combined random group structure and intra-group gradient segment aggregation mechanisms to prevent the exploitation of side-channel attacks. Like TrustFL, ShuffleFL also uses enclave to perform model aggregation.

### 4.3 Secure Multiparty Learning

This section turns to review and compare TEE-based protocols towards secure multi-party training.

**4.3.1 Problem Statement.** Secure multi-party training (SMT) enables  $n$  ( $n \geq 2$ ) parties holding data jointly performing a training task  $(\Phi, X_1, \dots, X_N)$  and generating a trained model  $M$ , where  $X_i$  is the  $i$ -th party's training data. The *correctness* of SMT means that the trained model is equal to the model that was trained on an honest party over the same data set. The *privacy* means that any party fails to learn any sensitive information about other parties' data. The *integrity* enforces each party performing  $\Psi_i(x_i)$  honestly and generating the operation integrity proof  $\Gamma_i$ .

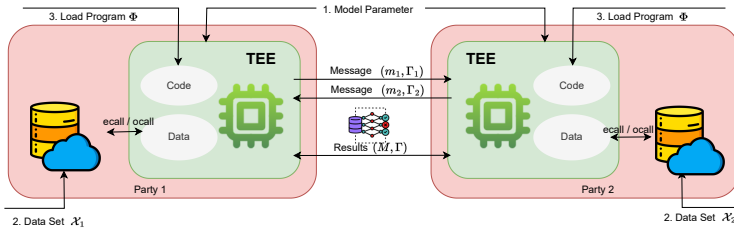


Fig. 9. Secure Multiparty Training

Fig. 9 shows a general workflow of secure two party training between two TEE-equipped parties, which can be easily extended to the multi-party setting. The two party builds a secure channel via remote attestation, after which all messages are communicated over the secure channel. Then three steps are followed. (1) Both enclave read the model parameter into TEE. (2) Both parties take their own data  $X_1$  and  $X_2$  as input, respectively. (3) Both parties ensure the training program  $\Phi$  is loaded into the TEEs successfully. After that the protocol executes interactively between the two parties. In each interaction, a message associated with a proof  $(m_i, \Gamma_i)$  is sent to another party. Here  $m_i$  is a committed message to ensure the privacy of the execution, and  $\Gamma_i$  is employed to prove the message  $m_i$  is computed honestly. The protocol halts until generating a convergent model and outputs a final trained model  $(M, \Gamma)$ , where  $\Gamma$  proves the protocol is run honestly.

**4.3.2 Research Status.** Although secure multi-party computation (MPC) is successfully applied to secure machine learning [29], only few solutions of multi-party machine learning training are designed without requiring a third party (or master node) to participate in computations. One possible explanation is that the training algorithms faces with complex computations, which affects the performance significantly. In [120], to reduce the communication overhead of MPC-based methods for privacy-preserving neural networks training, Ren et al. presented an  $n$ -party training framework for Graph Neural Networks (GNNs) using SGX. During training,  $n$  parties hold training data, and SGX shares training data by additive secret sharing and perform training operations on the local enclave. Following that, the party aggregates other  $n - 1$  parties' training results to obtain the trained model. The solution of [120] can output a trained GNN within dozens of seconds.

Table 2. Comparisons of Secure Machine Learning

Schemes	Settings		Methodology		Security					Performance	
	Framework	Function	Technologies	Primitives	Security Goals			Adversarial Models	Formal proof	Configurations	Costs
					Privacy	Integrity	Freshness				
[OMPML [106], USENIX Security, 2016]	SOT	Training	TEE+OP	AES-GCM, GPO	IO, exe, pattern	IO, exe	✓	Malicious Server/Client	✗	Dataset: MNIST, SUSY, MovieLens, Nursery Models: K-Means, CNN, SVM, Matrix factorization Baseline: MPC	1.7-2.5x speedup
[PRIVADO [58], arXiv, 2018]	SOI	Inference	TEE	DO,TLS	IO, exe	IO, exe	✗	Malicious Server	✗	Dataset: MNIST, CIFAR10 Models: MLP, LeNet, VGG19, Widerresnet, ResNet29/50/110, AlexNet, SqueezeNet, InceptionV3, DenseNet Platform: Intel SGX SDK, torch Baseline: Plaintext Model	0.18x slowdown
[Chiron [68], arXiv, 2018]	SDT	Training	TEE	AES-GCM, DF, TLS	IO, exe	IO, exe	✗	Malicious Server	✗	Dataset: CIFAR-10, ImageNet Models: VGG-9, AlexNet Platforms: Intel SGX, Ryoan sandbox Baseline: Plaintext Model	0.04-0.2x slowdown
[DeepEnclave [59], arXiv, 2018]	SOI	Inference	TEE	AE, TLS	IO, exe	IO, exe	✗	Malicious Server	✓	Dataset: ImageNet Models: Darknet, Extraction, DeseNets Baseline: Plaintext Model	1.64-2.54x slowdown
[SLALOM [142], ICLR, 2019]	SOI	Inference	TEE	Frei., SC PRNG	IO (model, input)	exe	✗	Malicious Server	✓	Dataset: ImageNet Models: VGG16, MobileNet Platforms: Intel SGX, TensorFlow Baseline: Fully in Enclave	4-20x speedup
[TENSORSCONE [81], arXiv, 2019]	SOT	Training	TEE	TLS	IO, exe	IO, exe	✓	Malicious Server	✗	Dataset: CIFAR-10 Models: InceptionV4 Platforms: Intel SGX, SCONE, TensorFlow Baseline: Plaintext Model	0.8-0.9x slowdown
[OCCUMENCY [84], MobiCom, 2019]	SOI	Inference	TEE	AES, TLS	IO, exe	IO, exe	✗	Malicious Server	✗	Dataset: ImageNet Models: AlexNet, GoogleNet, ResNet50/101/152, VGG16/19, YOLO Platforms: Intel SGX, Caffe	Fully in Enclave: 3.6x speedup Plaintext Model: 0.72x slowdown
[Origami [100], arXiv, 2019]	SOI	Inference	TEE	-	IO	-	✗	Malicious Server	✗	Dataset: ImageNet Models: VGG16/19 Baseline: Slalom	11-15.1x speedup
[Telekine [67], NSDI, 2020]	SOT, SOI	Training Inference	GPU TEE+OP	AES-GCM, MAC, LTS	IO, pattern	IO, exe	✓	Malicious Server	✗	Dataset: ImageNet Models: ResNet, InceptionV3, DenseNet Platforms: AMD's ROCm 1.8, MXNet Baseline: Plaintext Model	Training: 1.08-1.23x slowdown Inference: 1.0-10x slowdown
[eNNclave [125], AISec, 2020]	SOI	Inference	TEE	AES-GCM	IO	IO	✗	Malicious Server/Client	✗	Dataset: MIT67, Flowers, Amazon review data Models: VGG-16, VGG-19, CNN Baseline: Plaintext Model	17x-65x slowdown
[VESSELS [74], SoCC, 2020]	SOI	Inference	TEE	-	IO, exe	IO, exe	✗	Malicious Server	✗	Dataset: ImageNet Models: AlexNet, ResNet101, ResNet152, DenseNet201, ResNext152, DarkNet53, InceptionV3, VGG16, YoloV3 Baseline: Plaintext Model	Inference: 5.01-20.9x slowdown
[ShadowNet [140], arXiv, 2020]	SDT	Training Inference	TEE	-	Model	-	✗	Malicious Server/Client	✓	Dataset: ImageNet, CIFAR-10 Models: MobileNets, ResNet-44, AlexNet, MiniVGG Platforms: TensorFlow Lite Baseline: Fully in Enclave	0.61-2.21 speedup
[TrustFL [160], INFOCOM, 2020]	SDT	Training	TEE	PRF, HMAC	IO, exe	IO, exe	✗	Semi-honest Server Dishonest Participant	✗	Dataset: CIFAR10, ImageNet Models: VGG16/19, ResNet Baseline: Fully in Enclave	10-100x speedup
[DarkneTZ [98], MobiSys, 2020]	SDT	Training Inference	TEE	AES-GCM	IO, exe	IO, exe	✗	Malicious Client	✗	Dataset: CIFAR100, ImageNet Tiny Models: AlexNet, VGG7 Platforms: ARM TrustZone Baseline: Plaintext Model	0.03-0.1x slowdown

**GPO:** General-purpose Oblivious, **DO:** Data Oblivious, **Frei.:** Freivalds' algorithm, **SC:** Stream ciphers, **PRNG:** Pseudo Random Number Generator, **AE:** Authenticated Encryption, **MPC:** Multiple Party Computation

Schemes	Settings		Methodology		Security					Performance	
	Framework	Function	Technologies	Primitives	Security Goals			Adversarial Models	Formal proof	Configurations	Costs
					Privacy	Integrity	Freshness				
[SECURETF [119], Middleware, 2020]	SDT	Training Inference	TEE	ECDH, TLS	IO, exe	IO, exe	✗	Malicious Server	✗	Dataset: CIFAR10, MNIST Models: Inception v3/v4, DeseNet Platforms: Intel SGX, TensorFlow Baseline: Plaintext Model	14x slowdown
[AegisDNN [152], RTSS, 2021]	SOI	Inference	TEE	-	IO, exe	IO, exe	✗	Malicious Server	✗	Dataset: - Models: lenet, Alecnet, ResNet-18, Pilotnet Platforms: Intel SGX, TensorFlow, Caffe, Pytorch Baseline: Plaintext Model	0.8-1.2 slowdown
[AsymML [103], arXiv, 2021]	SOT, SOI	Training Inference	TEE	-	IO (model, input)	-	✗	Malicious Server	✗	Dataset: ImageNet, CIFAR-10 Models: VGG16/19, ResNet18/34 Baseline: Fully in Enclave	Training: 11.2x speedup Inference: 7.6x speedup
[SEAR [163], TDSC, 2021]	SDT	Training	TEE	AES-GCM, Elliptic Curve	IO, exe	IO, exe	✗	Semi-honest Server/Client	✗	Dataset: MNIST, CIFAR-10 Models: CNN, ResNet56 Baseline: ASS-based solution Dataset: MNIST, CIFAR-10	4-6x speedup
[SecDeep [90], IoTDI, 2021]	SDT	Training Inference	TEE	AES-CTR, MD5	IO, exe	IO, exe	✗	Malicious Server	✗	Models: SqueezeNet, MobileNet V1/V2, GoogleNet, Yolo Tiny, ResNet50, Inception BN Platforms: HiKey960 board, ARM TrustZone Baseline: Plaintext Model	16-172x slowdown
[Citadel [158], SoCC, 2021]	SDT	Training	TEE	AES-CBC-256 ZSM, TSH	IO, exe	IO, exe	✗	Semi-honest Server/Client	✗	Models: AlexNet, AlexNetL, SpamNet, MNIST Platforms: Intel SGX, TensorFlow Baseline: Fully in Enclave	Training: 1.09-1.73x slowdown
[Mlcapsule [62], CVPR, 2021]	SDT	Training Inference	TEE	-	IO, exe	IO, exe	✗	Malicious Client	✗	Dataset: CIFAR100 Models: VGG-16 Baseline: Plaintext Model	Inference: 1.2-3x overhead
[StarFL [66], ACM TIST, 2021]	SDT	Training	TEE	AES-GCM	IO, exe	IO, exe	✗	Malicious Client	✗	Dataset: CIFAR100, ImageNet Tiny Models: AlexNet, VGG7 Platforms: ARM TrustZone Baseline: Plaintext Model	less than 5% slowdown
[Model Protection [65], TDSC, 2021]	SDT	Training Inference	TEE	-	Model Privacy	IO, exe	✗	Honest Server Semi-honest Client	✗	Dataset: CIFAR10/100, ImageNet Models: InceptionResNetV2, VGG19, MobileNetV2, NASNet Platform: Intel SGX, TensorFlow Baseline: DarknetZ	2.2-7.8x speedup
[SecureFL [82], SEC, 2021]	SDT	Training Inference	TEE	AES	IO, exe	IO, exe	✗	Semi-honest Server/Client	✗	Dataset: MNIST, CIFAR10, Tiny ImageNet Models: LeNet, VGG7/16 Platform: Intel SGX, Raspberry Pi 3B+ Baseline: Plaintext Model	0.1-0.3x slowdown
[Darknight [63], MICRO, 2021]	SOT, SOI	Training Inference	TEE	MM	IO, exe	IO, exe	✗	Malicious Server	✓	Dataset: CIFAR10, ImageNet Models: VGG16, ResNet50, MobileNetV2 Platform: Intel SGX, GTX1080 Ti Baseline: Fully in Enclave	Training: 6.5x speedup Inference: 12.5x speedup
[Plinius [157], DSN, 2021]	SOT, SOI	Training Inference	TEE	AES-GCM, PM	IO, exe	IO, exe	✗	Malicious Server	✗	Dataset: MNIST Models: CNN Platform: Intel SGX SDK Baseline: Fully in Enclave	2.5-3.5x Speedup
[Goten [102], AAAI, 2021]	SOT, SOI	Training Inference	TEE	ASS	IO, exe	IO, exe	✗	Semi-honest server	✗	Dataset: CIFAR10 Models: VGG11 Baseline: CaffeScone, Falcon	6.84-132.64x speedup

**ECDH**: Elliptic Curve Diffie-Hellman, **ZSM**: Zero-sum Masking, **TSH**: Tree-structured Hierarchical, **ASS**: Additive Secret Sharing, **MM**: Matrix Masking;

Schemes	Settings		Methodology		Security					Performance	
					Security Goals			Adversarial Models	Formal proof		
	Framework	Function	Technologies	Primitives	Privacy	Integrity	Freshness			Configurations	Costs
[PPFL [97], MobiSys, 2021]	SDT	Training	TEE	AES-GCM	IO, exe	IO, exe	✗	Semi-honest Server/Client	✗	Dataset: MNIST, CIFAR10 Models: LeNet, AlexNet, VGG19 Platform: ARM TrustZone Baseline: Fully in Enclave	1.5-3x slowdown
[PERUN [107], DBSec, 2021]	SOT, SOI	Training Inference	TEE	-	IO, exe	IO, exe	✗	Semi-honest	✗	Dataset: CIFAR10, Medical dataset Models: 2D-U-Net Platform: TensowFlow, Intel SGX	Plaintext Model: 0.96x slowdown secureTF : 1560x
[SOTERIA [23], ePrint, 2021]	SDT	Training	TEE	AES-GCM	IO, exe	IO, exe	✗	Malicious Server	✗	Dataset: no given Models: ALS, PCA, GBT, LR, Naive Bayes, LDA, K-means Platform: Intel SGX SDK, Aparch spark Baseline: Fully in Enclave	1.7x speedup
[FAML [111], WWW, 2022]	SOI	Inference	TEE	-	IO	IO	✗	Malicious Server	✗	Dataset: Adult, Bank, COMPAS, German, LSAC Models: LR, SVM, NN Baseline: Plaintext Model	1.18-2.43x slowdown
[AsymML [104], PETS, 2022]	SOT	Training	TEE	DP	IO, exe	IO, exe	✗	Malicious Server	✓	Dataset: CIFAR10, ImageNet Models: VGG16/19, ResNet18/34 Platforms: Intel SGX, Pytorch Baseline: Fully in Enclave	5.8-7.6x speedup
[GINN, [5], CODASPY, 2022]	SOT	Training	TEE	SKE/PKE SHA256	IO, exe	IO, exe	✗	Malicious Server	✓	Dataset: MNIST, CIFAR10 Models: VGG16/19, ResNet152, ResNet34s Baseline: Fully in Enclave	2x-20x speedup

**PM:** Persistent Memory, **DP:** Differential Privacy, **SKE:** Symmetric Encryption, **PKE:** Public Key Encryption;



Besides, CryptFlow [80] leverages secret sharing to distribute the sensitive input data to several servers and then achieves correctness and confidentiality by cryptographic primitives and integrity by TEE technologies. CryptFlow views the inference as one iteration of training, therefore their method is also suitable for secure multiparty training.

#### 4.4 Comparisons

A comprehensive comparison is made in Table 2 to compare existing secure computation protocols for machine learning tasks (including training and inference) with regard to our assessment criteria. It is not hard to employ TEE to achieve basic confidentiality and integrity, thus most of the protocols are designed against malicious parties. However, the references against side-channel attacks and replay attacks are very scarce, except for [67, 106]. Besides, the references [5, 59, 63, 104, 140, 142] also present formal proof security analysis for their proposals. As for performance, we observe that most of the references compare their protocol with either the plaintext or the fully in enclave baseline.

### 5 CONCLUSIONS AND FUTURE DIRECTIONS

The Trusted Execution Environment (TEE) has become increasingly popular for both academic research and industrial adoption to achieve secure computation with confidentiality and integrity guarantees. This survey provided a comprehensive overview of state-of-the-art TEE-enabled secure computation protocols. Rest on the proposed taxonomy and systematical assessment criteria, we conducted comprehensive comparisons on TEE-enabled secure computation for both general-purpose function and specific ones, including secure machine learning and secure database queries which have wide applications in practice. With the maturing and further development of TEE technologies, we envisage that TEE-enabled secure computation would become a mainstream for practical adoption in the near future. Below we discuss several remaining challenges and future research directions of this promising technology.

- *General-purpose computation on large-scale data.* As shown in the survey, existing solutions for general-purpose secure computation are only runnable on small-scale data. However, it is still very challenging to design general-purpose protocols for large-scale data sets. Although moderately weakening the security level to gain performance improvement is a viable strategy, balancing the trade-off between security and performance is known non-trivial task for a long time [43].
- *Incremental computation for real-time systems.* Since data is growing daily, it is more practical to perform secure computation based on the previously computed results and the new data. Incremental computation [99] takes the new data and the previous results as inputs to derive the updated results. We observe that many existing real-time systems employ the incremental algorithm to achieve real-time response. Employing TEE to design secure incremental computation protocols is very interesting and promising for securing real-time applications. Besides, the technique could also be applied to privacy preserving continual learning and reinforcement learning.
- *Mobile-friendly computation.* Globally, mobile devices are significantly outnumbering desktops and laptops. Although the computing power of mobile devices have kept being improved, they still have limitations in terms of capacity and resource, such as limited power supply and limited space for cooling, etc. Utilizing TEE in mobile computing [6, 26] and designing a mobile-friendly secure computation framework (e.g., offloading [79]) has not been well investigated and is worth exploring.

- *System protection against untrusted computation.* Secure computation assumes that TEE's host is malicious and thus achieves confidential computing on sensitive data. In contrast, the untrusted computation means that the program loaded into the TEE is destructive (e.g. Trojan Horse) and thus steals sensitive information from the host. Designing novel approaches to prevent TEE-based untrusted computation is another topic that demands further investigation [69, 110].
- *TEE trust management.* TEE provides the confidentiality and integrity guarantee of secure computation. However, in practice, not all parties fully trust the TEE. For instance, parties may worry that the TEE is comprised/controlled by another party and thus have no trust on TEE [150]. On the other hand, the minimal trust on TEE proposed in [80] means that secure computation protocols only make trust assumptions on the integrity protection of the hardware, and the confidential computing is shifted to the protocol design. Weakening or diversifying the trust on TEE is a promising direction for future research.

## ACKNOWLEDGMENTS

This research is supported by the AXA Research Fund and the National Natural Science Foundation of China, No. 62202358.

## REFERENCES

- [1] Joshua Allen, Bolin Ding, Janardhan Kulkarni, Harsha Nori, Olga Ohrimenko, and Sergey Yekhanin. 2019. An algorithmic framework for differentially private data analysis on trusted processors. *Advances in Neural Information Processing Systems, (NeurIPS)* 32 (2019).
- [2] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. 2013. Innovative technology for CPU based attestation and sealing. In *Proc. on hardware and architectural support for security and privacy, (HASP)*, Vol. 13.
- [3] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. 2014. Secure multiparty computations on bitcoin. In *IEEE Symposium on Security and Privacy, (S&P)*. 443–458.
- [4] Arvind Arasu, Ken Eguro, Manas Joglekar, Raghav Kaushik, Donald Kossmann, and Ravi Ramamurthy. 2015. Transaction processing on confidential data using cipherbase. In *International Conference on Data Engineering, (ICDE)*. 435–446.
- [5] Aref Asvadihirehjini, Murat Kantarcioglu, and Bradley Malin. 2022. GINN: Fast GPU-TEE Based Integrity for Neural Network Training. In *Proceedings of the Twelfth ACM Conference on Data and Application Security and Privacy*. 4–15.
- [6] Ahmad Atamli-Reineh, Ravishankar Borgaonkar, Ranjbar A Balisane, Giuseppe Petracca, and Andrew Martin. 2016. Analysis of trusted execution environment usage in samsung KNOX. In *Proc. of Workshop on System Software for Trusted Execution*. 1–6.
- [7] Raad Bahmani, Manuel Barbosa, Ferdinand Brasser, Bernardo Portela, Ahmad-Reza Sadeghi, Guillaume Scerri, and Bogdan Warinschi. 2017. Secure multiparty computation from SGX. In *International Conference on Financial Cryptography and Data Security, (FC)*. 477–497.
- [8] Maurice Bailleu, Dimitra Giantsidi, Vasilis Gavrielatos, Vijay Nagarajan, Pramod Bhatotia, et al. 2021. Avocado: A Secure In-Memory Distributed Storage System. In *2021 USENIX Annual Technical Conference, (USENIX ATC)*. 65–79.
- [9] Maurice Bailleu, Jorg Thalheim, Pramod Bhatotia, Christof Fetzer, Michio Honda, and Kapil Vaswani. 2019. SPEICHER: Securing LSM-based Key-Value Stores using Shielded Execution. In *USENIX Conference on File and Storage Technologies, (FAST)*. 173–190.
- [10] Sven Balnojan. 2020. *The Future Of Good Data — What You Should Know Now!* Retrieved Oct 9, 2022 from <https://towardsdatascience.com/the-future-of-good-data-what-you-should-know-now-f2a312a0e469>
- [11] Feng Bao, Robert H Deng, and Wenbo Mao. 1998. Efficient and practical fair exchange protocols with off-line TTP. In *Proc. IEEE Symposium on Security and Privacy, (S&P)*. 77–85.
- [12] Manuel Barbosa, Bernardo Portela, Guillaume Scerri, and Bogdan Warinschi. 2016. Foundations of hardware-based attested computation and application to SGX. In *IEEE European Symposium on Security and Privacy, (EuroS&P)*. 245–260.
- [13] Jones Bater, Gregory Elliott, Craig Eggen, Satyender Goel, Abel N Kho, and Jennie Rogers. 2017. SMCQL: Secure Query Processing for Private Data Networks. *Proc. VLDB Endow.* 10, 6 (2017), 673–684.
- [14] Jones Bater, Xi He, William Ehrich, Ashwin Machanavajjhala, and Jennie Rogers. 2018. Shrinkwrap: efficient sql query processing in differentially private data federations. *Proc. of the VLDB Endowment* 12, 3 (2018).

- [15] Andrew Baumann, Marcus Peinado, and Galen Hunt. 2015. Shielding applications from an untrusted cloud with haven. *ACM Transactions on Computer Systems, (TOCS)* 33, 3 (2015), 1–26.
- [16] Mihir Bellare, Marc Fischlin, Shafi Goldwasser, and Silvio Micali. 2001. Identification protocols secure against reset attacks. In *Proc. of EUROCRYPT*. 495–511.
- [17] Fabrice Benhamouda, Geoffroy Couteau, David Pointcheval, and Hoeteck Wee. 2015. Implicit zero-knowledge arguments and applications to the malicious setting. In *Annual Cryptology Conference, (CRYPTO)*. 107–129.
- [18] Iddo Bentov, Yan Ji, Fan Zhang, Lorenz Breidenbach, Philip Daian, and Ari Juels. 2019. Tesseract: Real-time cryptocurrency exchange using trusted hardware. In *Proc. of ACM SIGSAC Conference on Computer and Communications Security, (CCS)*. 1521–1538.
- [19] Manuel Blum. 1983. How to exchange (secret) keys. *ACM Transactions on computer systems, (TOCS)* 1, 2 (1983), 175–193.
- [20] Dan Bogdanov, Sven Laur, and Jan Willemson. 2008. Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security, (ESORICS)*. 192–206.
- [21] Amine Boulemtafes, Abdelouahid Derhab, and Yacine Challal. 2020. A review of privacy-preserving techniques for deep learning. *Neurocomputing* 384 (2020), 21–45.
- [22] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. 2020. Zexe: Enabling decentralized private computation. In *IEEE Symposium on Security and Privacy, (S&P)*. 947–964.
- [23] Cláudia Brito, Pedro Ferreira, Bernardo Portela, Rui Oliveira, and João Paulo. 2021. Soteria: Privacy-Preserving Machine Learning for Apache Spark. *Cryptology ePrint Archive* (2021).
- [24] Niklas Büscher, Daniel Demmler, Stefan Katzenbeisser, David Kretzmer, and Thomas Schneider. 2018. HyCC: Compilation of hybrid protocols for practical secure computation. In *Proc. of ACM SIGSAC Conference on Computer and Communications Security, (CCS)*. 847–861.
- [25] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. 2007. Universally composable security with global setup. In *Theory of Cryptography Conference, (TCC)*. 61–85.
- [26] Dhiman Chakraborty, Lucjan Hanzlik, and Sven Bugiel. 2019. simTPM: User-centric TPM for Mobile Devices. In *USENIX Security Symposium*. 533–550.
- [27] David Champagne and Ruby B Lee. 2010. Scalable architectural support for trusted software. In *The International Symposium on High-Performance Computer Architecture, (HPCA)*. 1–12.
- [28] Zhao Chang, Dong Xie, Sheng Wang, and Feifei Li. 2022. Towards Practical Oblivious Join. (2022).
- [29] Joseph I Choi and Kevin RB Butler. 2019. Secure multiparty computation and trusted hardware: Examining adoption challenges and opportunities. *Security and Communication Networks* (2019).
- [30] Joseph I Choi, Dave Tian, Grant Hernandez, Christopher Patton, Benjamin Mood, Thomas Shrimpton, Kevin RB Butler, and Patrick Traynor. 2019. A hybrid approach to secure function evaluation using SGX. In *Proc. of ACM Asia Conference on Computer and Communications Security, (AsiaCCS)*. 100–113.
- [31] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kapchuk, and Ian Miers. 2017. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In *Proc. of ACM SIGSAC Conference on Computer and Communications Security, (CCS)*. 719–728.
- [32] Richard Cleve. 1986. Limits on the security of coin flips when half the processors are faulty. In *Proc. ACM symposium on Theory of computing, (STOC)*. 364–369.
- [33] Cláudio Correia, Miguel Correia, and Luís Rodrigues. 2020. Omega: a secure event ordering service for the edge. In *Annual IEEE/IFIP International Conference on Dependable Systems and Networks, (DSN)*. 489–501.
- [34] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained. *Manuscript* (2016).
- [35] Victor Costan, Ilia Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal hardware extensions for strong software isolation. In *USENIX Security Symposium*. 857–874.
- [36] Ronald Cramer, Ivan Damgård, and Ueli Maurer. 2000. General secure multi-party computation from any linear secret-sharing scheme. In *Proc. of EUROCRYPT*. 316–334.
- [37] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P Smart. 2013. Practical covertly secure MPC for dishonest majority—or: breaking the SPDZ limits. In *European Symposium on Research in Computer Security, (ESORICS)*. 1–18.
- [38] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. 2012. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference, (CRYPTO)*. 643–662.
- [39] Hung Dang, Tien Tuan Anh Dinh, Ee-Chien Chang, and Beng Chin Ooi. 2017. Privacy-Preserving Computation with Trusted Computing via Scramble-then-Compute. *Proc. Priv. Enhancing Technol. (PET)* 2017, 3 (2017), 21.
- [40] Hung Dang, Dat Le Tien, and Ee-Chien Chang. 2019. Towards a marketplace for secure outsourced computations. In *European Symposium on Research in Computer Security*. 790–808.
- [41] Ankur Dave, Chester Leung, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2020. Oblivious cooperative analytics using hardware enclaves. In *Proc. of the European Conference on Computer Systems*. 1–17.

- [42] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY-A framework for efficient mixed-protocol secure two-party computation.. In *Network and Distributed System Security, (NDSS)*.
- [43] Daniel Deutch, Ariel Frankenthal, Amir Gilad, and Yuval Moskovitch. 2021. On optimizing the trade-off between privacy and utility in data provenance. In *Proc. of International Conference on Management of Data, (SIGMOD)*. 379–391.
- [44] Shuai Ding and Torsten Suel. 2011. Faster top-k document retrieval using block-max indexes. In *Proc. ACM conference on Research and development in Information Retrieval, (SIGIR)*. 993–1002.
- [45] Tien Tuan Anh Dinh, Prateek Saxena, Ee-Chien Chang, Beng Chin Ooi, and Chunwang Zhang. 2015. M2R: Enabling Stronger Privacy in MapReduce Computation. In *USENIX Security Symposium*. 447–462.
- [46] Jack Doerner and Abhi Shelat. 2017. Scaling ORAM for secure computation. In *Proc. of ACM SIGSAC Conference on Computer and Communications Security, (CCS)*. 523–535.
- [47] Huayi Duan, Cong Wang, Xingliang Yuan, Yajin Zhou, Qian Wang, and Kui Ren. 2019. LightBox: Full-stack protected stateful middlebox at lightning speed. In *Proc. of the ACM SIGSAC Conference on Computer and Communications Security, (CCS)*. 2351–2367.
- [48] Kha Dinh Duy, Taehyun Noh, Siwon Huh, and Hojoon Lee. 2021. Confidential Machine Learning Computation in Untrusted Environments: A Systems Security Perspective. *IEEE Access* 9 (2021), 168656–168677.
- [49] Saba Eskandarian and Matei Zaharia. 2019. OblIDB: oblivious query processing for secure databases. *Proc. of the VLDB Endowment* 13, 2 (2019), 169–183.
- [50] Shufan Fei, Zheng Yan, Wenxiu Ding, and Haomeng Xie. 2021. Security vulnerabilities of SGX and countermeasures: A survey. *ACM Computing Surveys, (CSUR)* 54, 6 (2021), 1–36.
- [51] Susanne Felsen, Ágnes Kiss, Thomas Schneider, and Christian Weinert. 2019. Secure and private function evaluation with Intel SGX. In *Proc. of ACM SIGSAC Conference on Cloud Computing Security Workshop, (CCSW)*. 165–181.
- [52] Bernardo Ferreira, Bernardo Portela, Tiago Oliveira, Guilherme Borges, Henrique Joao Domingos, and Joao Leita. 2020. Boolean searchable symmetric encryption with filters on trusted hardware. *IEEE Transactions on Dependable and Secure Computing* (2020). <https://doi.org/10.1109/TDSC.2020.3012100>
- [53] Benny Fuhry, Raad Bahmani, Ferdinand Brasser, Florian Hahn, Florian Kerschbaum, and Ahmad-Reza Sadeghi. 2017. HardIDX: Practical and secure index with SGX. In *IFIP Annual Conference on Data and Applications Security and Privacy, (DBSec)*. 386–408.
- [54] William Futral and James Greene. 2013. Fundamental Principles of Intel® TXT. In *Intel® Trusted Execution Technology for Server Platforms*. 15–36.
- [55] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proc. of ACM symposium on Theory of computing, (STOC)*. 169–178.
- [56] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Proc. of ACM symposium on Theory of computing, (STOC)*. 307–328.
- [57] Oded Goldreich and Rafail Ostrovsky. 1996. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)* 43, 3 (1996), 431–473.
- [58] Karan Grover, Shruti Tople, Shweta Shinde, Ranjita Bhagwan, and Ramachandran Ramjee. 2018. Privado: Practical and secure DNN inference with enclaves. *arXiv preprint:1810.00602* (2018).
- [59] Zhongshu Gu, Heqing Huang, Jialong Zhang, Dong Su, Ankita Lamba, Dimitrios Pendarakis, and Ian Molloy. 2018. Securing input data of deep learning inference systems via partitioned enclave execution. *arXiv preprint arXiv:1807.00969* (2018).
- [60] Debayan Gupta, Benjamin Mood, Joan Feigenbaum, Kevin Butler, and Patrick Traynor. 2016. Using intel software guard extensions for efficient two-party secure function evaluation. In *International Conference on Financial Cryptography and Data Security, (FC)*. 302–318.
- [61] Joseph Halpern and Vanessa Teague. 2004. Rational secret sharing and multiparty computation. In *Proc. of ACM symposium on Theory of computing, (STOC)*. 623–632.
- [62] Lucjan Hanzlik, Yang Zhang, Kathrin Grosse, Ahmed Salem, Maximilian Augustin, Michael Backes, and Mario Fritz. 2021. Mlcapsule: Guarded offline deployment of machine learning as a service. In *Proc. of IEEE/CVF Conference on Computer Vision and Pattern Recognition, (CVPR)*. 3300–3309.
- [63] Hanieh Hashemi, Yongqin Wang, and Murali Annavaram. 2021. DarKnight: An accelerated framework for privacy and integrity preserving deep learning using trusted hardware. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 212–224.
- [64] Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Rebecca N Wright. 2018. Privacy-preserving machine learning as a service. *Proc. Priv. Enhancing Technol.* 2018, 3 (2018), 123–142.
- [65] Jiahui Hou, Huiqi Liu, Yunxin Liu, Yu Wang, Peng-Jun Wan, and Xiang-Yang Li. 2021. Model Protection: Real-time Privacy-preserving Inference Service for Model Privacy at the Edge. *IEEE Transactions on Dependable and Secure Computing* (2021).

- [66] Anbu Huang, Yang Liu, Tianjian Chen, Yongkai Zhou, Quan Sun, Hongfeng Chai, and Qiang Yang. 2021. StarFL: Hybrid federated learning architecture for smart urban computing. *ACM Transactions on Intelligent Systems and Technology (TIST)* 12, 4 (2021), 1–23.
- [67] Tyler Hunt, Zhipeng Jia, Vance Miller, Ariel Szekely, Yige Hu, Christopher J Rossbach, and Emmett Witchel. 2020. Telekine: Secure Computing with Cloud GPUs. In *USENIX Symposium on Networked Systems Design and Implementation, (NSDI)*. 817–833.
- [68] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. 2018. Chiron: Privacy-preserving machine learning as a service. *arXiv preprint:1803.05961* (2018).
- [69] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. 2016. Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data. In *USENIX Symposium on Operating Systems Design and Implementation, (OSDI)*. 533–549.
- [70] Charlie Jaccome, Steve Kremer, and Guillaume Scerri. 2017. Symbolic models for isolated execution environments. In *IEEE European Symposium on Security and Privacy, (EuroS&P)*. 530–545.
- [71] David Kaplan, Jeremy Powell, and Tom Woller. 2016. AMD memory encryption. *White paper* (2016).
- [72] Gabriel Kapthuk, Ian Miers, and Matthew Green. 2017. Giving state to the stateless: Augmenting trustworthy computation with ledgers. *Network and Distributed System Security, (NDSS)* (2017).
- [73] Fumiyuki Kato, Yang Cao, and Mastoshi Yoshikawa. 2021. PCT-TEE: Trajectory-based Private Contact Tracing System with Trusted Execution Environment. *ACM Transactions on Spatial Algorithms and Systems, (TSAS)* 8, 2 (2021), 1–35.
- [74] Kyungtae Kim, Chung Hwan Kim, Junghwan" John" Rhee, Xiao Yu, Haifeng Chen, Dave Tian, and Byoungyoung Lee. 2020. Vessels: efficient and scalable deep learning prediction on trusted processors. In *Proc. of the 11th ACM Symposium on Cloud Computing*. 462–476.
- [75] Taehoon Kim, Joongun Park, Jaewook Woo, Seungheun Jeon, and Jaehyuk Huh. 2019. Shieldstore: Shielded in-memory key-value storage with sgx. In *Proc. of EuroSys*. 1–15.
- [76] Vladimir Kolesnikov and Ranjit Kumaresan. 2013. Improved OT extension for transferring short secrets. In *Annual Cryptology Conference, (CRYPTO)*. 54–70.
- [77] Simeon Krastnikov, Florian Kerschbaum, and Douglas Stebila. 2021. Efficient Oblivious Database Joins. *Proc. of the VLDB Endowment* 13, 11 (2021).
- [78] Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. 2012. Billion-Gate Secure Computation with Malicious Adversaries. In *21st USENIX Security Symposium, (USENIX Security)*. 285–300.
- [79] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava. 2013. A survey of computation offloading for mobile systems. *Mobile networks and Applications* 18, 1 (2013), 129–140.
- [80] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. Cryptflow: Secure tensorflow inference. In *IEEE Symposium on Security and Privacy, (S&P)*. 336–353.
- [81] Roland Kunkel, Do Le Quoc, Franz Gregor, Sergei Arnaudov, Pramod Bhatotia, and Christof Fetzer. 2019. Tensorscone: A secure tensorflow framework using intel sgx. *arXiv preprint:1902.04413* (2019).
- [82] Eugene Kuznetsov, Yitao Chen, and Ming Zhao. 2021. SecureFL: Privacy preserving federated learning with sgx and trustzone. In *2021 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 55–67.
- [83] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. 2020. Keystone: An open framework for architecting trusted execution environments. In *Proc. of European Conference on Computer Systems, (EuroSys)*. 1–16.
- [84] Taeyeon Lee, Zhiqi Lin, Saumay Pushp, Caihua Li, Yunxin Liu, Youngki Lee, Fengyuan Xu, Chenren Xu, Lintao Zhang, and Junehwa Song. 2019. Occlumency: Privacy-preserving remote deep-learning inference using SGX. In *Annual International Conference on Mobile Computing and Networking, (MobiCom)*. 1–17.
- [85] David Lie, Chandramohan Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John Mitchell, and Mark Horowitz. 2000. Architectural support for copy and tamper resistant software. *ACM SIGPLAN Notices* 35, 11 (2000), 168–177.
- [86] ARM Limited. 2009. *ARM Security Technology-Building a Secure System using TrustZone Technology*. Technical Report.
- [87] Joshua Lind, Oded Naor, Ittay Eyal, Florian Kelbert, Peter Pietzuch, and Emin Gün Sirer. 2018. Teechain: Reducing storage costs on the blockchain with offline payment channels. In *Proc. of ACM International Systems and Storage Conference, (SYSTOR)*. 125–125.
- [88] Yehuda Lindell and Benny Pinkas. 2007. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Annual Cryptology Conference, (CRYPTO)*. 52–78.
- [89] Chang Liu, Yan Huang, Elaine Shi, Jonathan Katz, and Michael Hicks. 2014. Automating efficient RAM-model secure computation. In *IEEE Symposium on Security and Privacy, (S&P)*. 623–638.
- [90] Renju Liu, Luis Garcia, Zaoxing Liu, Botong Ou, and Mani Srivastava. 2021. SecDeep: Secure and Performant On-device Deep Learning Inference Framework for Mobile and IoT Devices. In *Proceedings of the International Conference on Internet-of-Things Design and Implementation*. 67–79.

- [91] Michael Luby, Silvio Micali, and Charles Rackoff. 1983. How to simultaneously exchange a secret bit by flipping a symmetrically-biased coin. In *Annual Symposium on Foundations of Computer Science, (FOCS)*. 11–22.
- [92] Martin Maas, Eric Love, Emil Stefanov, Mohit Tiwari, Elaine Shi, Krste Asanovic, John Kubiatowicz, and Dawn Song. 2013. Phantom: Practical oblivious computation in a secure processor. In *Proc. of ACM SIGSAC conference on Computer & communications security, (CCS)*. 311–324.
- [93] Lior Malka. 2011. Vmccrypt: modular software architecture for scalable secure computation. In *Proc. of ACM conference on Computer and communications security, (CCS)*. 715–724.
- [94] Lorenzo Martignoni, Pongsin Poosankam, Matei Zaharia, Jun Han, Stephen McCamant, Dawn Song, Vern Paxson, Adrian Perrig, Scott Shenker, and Ion Stoica. 2012. Cloud terminal: secure access to sensitive applications from untrusted systems. In *USENIX ATC*. 165–182.
- [95] Sahar Mazloom and S Dov Gordon. 2018. Secure computation with differentially private access patterns. In *Proc. of ACM SIGSAC Conference on Computer and Communications Security, (CCS)*. 490–507.
- [96] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. 2013. Innovative instructions and software model for isolated execution.. In *Proc. on hardware and architectural support for security and privacy, (HASP)*, Vol. 10.
- [97] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. 2021. PPFL: privacy-preserving federated learning with trusted execution environments. In *Proc. on Mobile Systems, Applications, and Services, (MobiSys)*. 94–108.
- [98] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. 2020. DarkneTZ: towards model privacy at the edge using trusted execution environments. In *Proc. of the International Conference on Mobile Systems, Applications, and Services, (MobiSys)*. 161–174.
- [99] Moni Naor, Omer Paneth, and Guy N Rothblum. 2019. Incrementally verifiable computation via incremental PCPs. In *Theory of Cryptography Conference*. 552–576.
- [100] Krishna Giri Narra, Zhifeng Lin, Yongqin Wang, Keshav Balasubramaniam, and Murali Annavaram. 2019. Privacy-preserving inference in machine learning services using trusted execution environments. *arXiv preprint arXiv:1912.03485* (2019).
- [101] Jer Shyuan Ng, Wei Yang Bryan Lim, Nguyen Cong Luong, Zehui Xiong, Alia Asheralieva, Dusit Niyato, Cyril Leung, and Chunyan Miao. 2020. A survey of coded distributed computing. *arXiv preprint:2008.09048* (2020).
- [102] Lucien KL Ng, Sherman SM Chow, Anna PY Woo, Donald PH Wong, and Yongjun Zhao. 2021. Goten: GPU-Outsourcing trusted execution of neural network training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 14876–14883.
- [103] Yue Niu, Ramy E Ali, and Salman Avestimehr. 2021. AsymML: An Asymmetric Decomposition Framework for Privacy-Preserving DNN Training and Inference. *arXiv preprint arXiv:2110.01229* (2021).
- [104] Yue Niu, Ramy E Ali, and Salman Avestimehr. 2022. 3LegRace: Privacy-Preserving DNN Training over TEEs and GPUs. In *Proceedings of privacy enhancing technologies*. 1–21.
- [105] Olga Ohrimenko, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Markulf Kohlweiss, and Divya Sharma. 2015. Observing and preventing leakage in MapReduce. In *Proc. of ACM SIGSAC Conference on Computer and Communications Security, (CCS)*. 1570–1581.
- [106] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. 2016. Oblivious Multi-Party Machine Learning on Trusted Processors. In *USENIX Security Symposium*. 619–636.
- [107] Wojciech Ozga, Do Le Quoc, and Christof Fetzer. 2021. Perun: Confidential Multi-stakeholder Machine Learning Framework with Hardware Acceleration Support. In *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 189–208.
- [108] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *Proc. of EUROCRYPT*. 223–238.
- [109] Heejin Park, Shuang Zhai, Long Lu, and Felix Xiaozhu Lin. 2019. StreamBox-TZ: Secure Stream Analytics at the Edge with TrustZone. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. 537–554.
- [110] Joongun Park, Naegyong Kang, Taehoon Kim, Youngjin Kwon, and Jaehyuk Huh. 2020. Nested enclave: supporting fine-grained hierarchical isolation with SGX. In *ACM/IEEE Annual International Symposium on Computer Architecture, (ISCA)*. 776–789.
- [111] Saerom Park, Seongmin Kim, and Yeon-sup Lim. 2022. Fairness Audit of Machine Learning Models with Confidential Computing. In *Proceedings of the ACM Web Conference 2022*. 3488–3499.
- [112] Rafael Pass, Elaine Shi, and Florian Tramer. 2017. Formal abstractions for attested execution secure processors. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques, (EUROCRYPT)*. 260–289.
- [113] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. 2021. ABY2. 0: Improved Mixed-Protocol Secure Two-Party Computation. In *USENIX Security Symposium*. 2165–2182.

- [114] Souradyuti Paul and Ananya Shrivastava. 2019. Efficient fair multiparty protocols using Blockchain and trusted hardware. In *International Conference on Cryptology and Information Security in Latin America*. 301–320.
- [115] Global Platform. 2013. Global platform made simple guide: Trusted execution environment (tee) guide. *Derniere visite* 12, 4 (2013).
- [116] Raluca Ada Popa, Catherine MS Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: protecting confidentiality with encrypted query processing. In *Proc. ACM Symposium on Operating Systems Principles*. 85–100.
- [117] Kevin Poulsen. 2001. DirecTV attacks hacked smart cards. *The Register* (2001).
- [118] Christian Priebe, Kapil Vaswani, and Manuel Costa. 2018. EnclaveDB: A secure database using SGX. In *Proc. IEEE Symposium on Security and Privacy, (S&P)*. 264–278.
- [119] Do Le Quoc, Franz Gregor, Sergei Arnautov, Roland Kunkel, Pramod Bhatotia, and Christof Fetzer. 2020. secureTF: a secure TensorFlow framework. In *Proceedings of the 21st International Middleware Conference*. 44–59.
- [120] Yixuan Ren, Yixin Jie, Qingtao Wang, Bingbing Zhang, Chi Zhang, and Lingbo Wei. 2021. A Hybrid Secure Computation Framework for Graph Neural Networks. In *Proc. on Privacy, Security and Trust, (PST)*. 1–6.
- [121] CharlesLin SukritKalra RishabhPoddar and Raluca Ada Popa Joe Hellerstein. 2014. Secure Federated Analytics. (2014).
- [122] Nuno Santos, Rodrigo Rodrigues, Krishna P Gummadi, and Stefan Saroiu. 2012. Policy-Sealed Data: A New Abstraction for Building Trusted Cloud Services. In *USENIX Security Symposium*. 175–188.
- [123] Sajin Sasy, Sergey Gorbunov, and Christopher W Fletcher. 2017. ZeroTRACE: Oblivious memory primitives from intel sgx. *Network and Distributed System Security, (NDSS)* (2017).
- [124] Sajin Sasy and Olga Ohrimenko. 2019. Oblivious sampling algorithms for private data analysis. *Advances in Neural Information Processing Systems* 32 (2019).
- [125] Alexander Schlögl and Rainer Böhme. 2020. eNNclave: offline inference with model confidentiality. In *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*. 93–104.
- [126] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: Trustworthy data analytics in the cloud using SGX. In *IEEE symposium on security and privacy, (S&P)*. 38–54.
- [127] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.
- [128] Zihao Shan, Kui Ren, Marina Blanton, and Cong Wang. 2018. Practical secure computation outsourcing: A survey. *ACM Computing Surveys (CSUR)* 51, 2 (2018), 1–40.
- [129] Jinjin Shao, Shiyu Ji, Alvin Oliver Glova, Yifan Qiao, Tao Yang, and Tim Sherwood. 2020. Index obfuscation for oblivious document retrieval in a trusted execution environment. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1345–1354.
- [130] Fahad Shaon, Murat Kantarcioglu, Zhiqiang Lin, and Latifur Khan. 2017. Sgx-bigmatrix: A practical encrypted data analytic framework with trusted processors. In *Proc. of ACM SIGSAC Conference on Computer and Communications Security, (CCS)*. 1211–1228.
- [131] Jatinder Singh, Jennifer Cobbe, Do Le Quoc, and Zahra Tarkhani. 2021. Enclaves in the Clouds. *Commun. ACM* 64, 5 (2021), 42–51.
- [132] Rohit Sinha, Sivanarayana Gaddam, and Ranjit Kumaresan. 2019. LucidiTEE: A TEE-Blockchain System for Policy-Compliant Multiparty Computation with Fairness. *Cryptology ePrint Archive* (2019).
- [133] Sergei Skorobogatov. 2016. The bumpy road towards iPhone 5c NAND mirroring. *arXiv preprint:1609.04327* (2016).
- [134] Smartcard 1988. NIST Special Publication 500-157. Smart Card Technology: New Methods for Computer Access Control. <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication500-157.pdf>.
- [135] Jeongseok Son, Griffin Prechter, Rishabh Poddar, Raluca Ada Popa, and Koushik Sen. 2021. ObliCheck: Efficient Verification of Oblivious Algorithms with Unobservable State. In *USENIX Security Symposium*.
- [136] Emil Stefanov, Marten Van Dijk, Elaine Shi, T-H Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. 2018. Path ORAM: an extremely simple oblivious RAM protocol. *Journal of the ACM (JACM)* 65, 4 (2018), 1–26.
- [137] Pramod Subramanyan, Rohit Sinha, Ilia Lebedev, Srinivas Devadas, and Sanjit A Seshia. 2017. A formal foundation for secure remote execution of enclaves. In *Proc. of ACM SIGSAC Conference on Computer and Communications Security, (CCS)*. 2435–2450.
- [138] Nancy Sumrall and Manny Novoa. 2003. Trusted computing group (TCG) and the TPM 1.2 specification. In *Intel Developer Forum*, Vol. 32.
- [139] Yuanyuan Sun, Sheng Wang, Huorong Li, and Feifei Li. 2021. Building enclave-native storage engines for practical encrypted databases. *Proc. of the VLDB Endowment* 14, 6 (2021), 1019–1032.
- [140] Zhichuang Sun, Ruimin Sun, Changming Liu, Amrita Roy Chowdhury, Somesh Jha, and Long Lu. 2020. ShadowNet: A secure and efficient system for on-device model inference. *arXiv preprint arXiv:2011.05905* (2020).
- [141] Sandeep Tamrakar, Jian Liu, Andrew Paverd, Jan-Erik Ekberg, Benny Pinkas, and N Asokan. 2017. The circle game: Scalable private membership test using trusted hardware. In *Proc. of ACM on Asia Conference on Computer and*

- Communications Security, (AsiaCCS)*. 31–44.
- [142] Florian Tramèr and Dan Boneh. 2019. Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.
  - [143] Thijs Veugen, Frank Blom, Sebastiaan JA de Hoogh, and Zekeriya Erkin. 2015. Secure comparison protocols in the semi-honest model. *IEEE Journal of Selected Topics in Signal Processing* 9, 7 (2015), 1217–1228.
  - [144] Dhinakaran Vinayagamurthy, Alexey Gribov, and Sergey Gorbunov. 2019. StealthDB: a Scalable Encrypted Database with Full SQL Query Support. *Proc. Priv. Enhancing Technol. (PETS)* 2019, 3 (2019), 370–388.
  - [145] Nikolaj Volgushev, Malte Schwarzkopf, Ben Getchell, Mayank Varia, Andrei Lapets, and Azer Bestavros. 2019. Conclave: secure multi-party computation on big data. In *Proc. of European Conference on Computer Systems, (EuroSys)*. 1–18.
  - [146] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. 2018. Graviton: Trusted Execution Environments on {GPUs}. In *USENIX Symposium on Operating Systems Design and Implementation, (OSDI)*. 681–696.
  - [147] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. 2021. FALCON: Honest-Majority Maliciously Secure Framework for Private Deep Learning. *Proceedings on Privacy Enhancing Technologies* 2021, 1 (2021), 188–208.
  - [148] Xiao Shaun Wang, Yan Huang, TH Hubert Chan, Abhi Shelat, and Elaine Shi. 2014. SCORAM: oblivious RAM for secure computation. In *Proc. of ACM SIGSAC Conference on Computer and Communications Security, (CCS)*. 191–202.
  - [149] Pengfei Wu, Qi Li, Jianting Ning, Xinyi Huang, and Wei Wu. 2021. Differentially Oblivious Data Analysis with Intel SGX: Design, Optimization, and Evaluation. *IEEE Transactions on Dependable and Secure Computing* (2021).
  - [150] Pengfei Wu, Jianting Ning, Jiaming Shen, Hongbing Wang, and Ee-Chien Chang. 2022. Hybrid Trust Multi-party Computation with Trusted Execution Environment. *Network and Distributed System Security, (NDSS)* (2022).
  - [151] Pengfei Wu, Qingni Shen, Robert H Deng, Ximeng Liu, Yinghui Zhang, and Zhonghai Wu. 2019. OblIDC: an SGX-based oblivious distributed computing framework with formal proof. In *Proc. of ACM Asia Conference on Computer and Communications Security, (AsiaCCS)*. 86–99.
  - [152] Yecheng Xiang, Yidi Wang, Hyunjong Choi, Mohsen Karimi, and Hyoseung Kim. 2021. AegisDNN: Dependable and Timely Execution of DNN Tasks with SGX. In *2021 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 68–81.
  - [153] Min Xu, Antonis Papadimitriou, Ariel Feldman, and Andreas Haeberlen. 2018. Using differential privacy to efficiently mitigate side channels in distributed analytics. In *Proc. of the European Workshop on Systems Security, (EuroSec)*. 1–6.
  - [154] Min Xu, Antonis Papadimitriou, Andreas Haeberlen, and Ariel Feldman. 2019. Hermetic: Privacy-preserving distributed analytics without (most) side channels. *Technical Report* (2019).
  - [155] Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets. In *Annual Symposium on Foundations of Computer Science, (FOCS)*. 162–167.
  - [156] Xuefei Yin, Yanming Zhu, and Jiankun Hu. 2021. A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions. *ACM Computing Surveys (CSUR)* 54, 6 (2021), 1–36.
  - [157] Peterson Yuhala, Pascal Felber, Valerio Schiavoni, and Alain Tchana. 2021. Plinius: Secure and persistent machine learning model training. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 52–62.
  - [158] Chengliang Zhang, Junzhe Xia, Baichen Yang, Huancheng Puyang, Wei Wang, Ruichuan Chen, Istemi Ekin Akkus, Paarijaat Aditya, and Feng Yan. 2021. Citadel: Protecting Data Privacy and Model Confidentiality for Collaborative Learning. In *Proc. of the ACM Symposium on Cloud Computing, (SoCC)*. 546–561.
  - [159] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. 2016. Town crier: An authenticated data feed for smart contracts. In *Proc. of ACM SIGSAC conference on computer and communications security, (CCS)*. 270–282.
  - [160] Xiaoli Zhang, Fengting Li, Zeyu Zhang, Qi Li, Cong Wang, and Jianping Wu. 2020. Enabling execution assurance of federated learning at untrusted participants. In *IEEE Conference on Computer Communications, (INFOCOM)*. 1877–1886.
  - [161] Yuhui Zhang, Zhiwei Wang, Jiangfeng Cao, Rui Hou, and Dan Meng. 2021. ShuffleFL: gradient-preserving federated learning using trusted execution environment. In *Proceedings of the 18th ACM International Conference on Computing Frontiers*. 161–168.
  - [162] Bowen Zhao, Jiaming Yuan, Ximeng Liu, Yongdong Wu, Hwee Hwa Pang, and Robert H Deng. 2022. SOCI: A Toolkit for Secure Outsourced Computation on Integers. *IEEE Transactions on Information Forensics and Security* (2022).
  - [163] Lingchen Zhao, Jianlin Jiang, Bo Feng, Qian Wang, Chao Shen, and Qi Li. 2021. SEAR: Secure and efficient aggregation for byzantine-robust federated learning. *IEEE Transactions on Dependable and Secure Computing* (2021).
  - [164] Wenting Zheng, Ankur Dave, Jethro G Beekman, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2017. Opaque: An oblivious and encrypted distributed analytics platform. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 283–298.
  - [165] Wenchao Zhou, Yifan Cai, Yanqing Peng, Sheng Wang, Ke Ma, and Feifei Li. 2021. Veridb: an SGX-based verifiable database. In *Proc. of the International Conference on Management of Data, (SIGMOD)*. 2182–2194.



## A SECURE DATABASE QUERIES USING TEES

In this section, we review the TEE-based secure computation for handling encrypted database queries. Unlike general-purpose secure computation, secure database query protocols aim to implement *data query tasks* with optimal performance. Over the past two decades, the notion of encrypted databases [116] has been proposed to enhance data availability while keeping data confidential. It not only provides a hardware and software infrastructure for securely storing sensitive data, such as personal records, financial documents, and government data, but also offers data query services to authorized users. As before, we divide secure database queries into three categories: secure outsourced queries, secure distributed queries, and secure multiparty queries.

### A.1 Secure Outsourced Queries

**A.1.1 Problem Statement.** Generally, a Secure Outsourced Queries (SOQ) protocol consists of three entities: a data owner, a client, and a cloud server. First, the data owner encrypts the original database  $\mathbf{d}$  and then uploads its encrypted form  $(\llbracket \mathbf{d} \rrbracket, \llbracket \Lambda \rrbracket)$  to the cloud server, in which  $\llbracket \mathbf{d} \rrbracket$  is the encryption of original data and  $\llbracket \Lambda \rrbracket$  is an encrypted index allowing server to provide query services efficiently. Once the database is uploaded, any authorized client can submit an encrypted query  $\llbracket q \rrbracket$  to the cloud server. The server performs the query services in encrypted form and obtains  $(\llbracket y \rrbracket, \Gamma) = \Phi((\llbracket \mathbf{d} \rrbracket, \llbracket \Lambda \rrbracket), \llbracket q \rrbracket)$ , where  $\Phi$  is a well-designed secure outsourced analytics protocol and  $y$  is the query result. Obviously, it is a special case of secure outsourced computation. The *correctness* means that the accurate query results are returned to the client, and the *completeness* means that all correct results should be returned. The *privacy* means that the cloud server cannot learn anything sensitive information about  $q$  (query privacy),  $(\mathbf{d}, \Lambda)$  (database privacy), and  $y$  (output privacy). The *integrity* forces the cloud server to run the program  $\Phi$  honestly, where  $\Gamma$  is the proof for proving the integrity of execution.

The early SOQ protocols [4, 116] are based on various cryptographic primitives, for example, searchable encryption, homomorphic encryption, structured encryption, order-preserving encryption for comparison-based queries, accumulator, and Merkle-hash proof for providing integrity guarantee. However, they just support some simple queries and fail to answer complex queries efficiently, such as queries involving join operations. Since the emergence of trusted hardware, TEE-based SOQ has provided a promising way to achieve a practical encrypted database while providing privacy protection and integrity guarantee. Besides, TEE is also a natural choice for implementing complex access-control policies in the encrypted database.

As shown in Fig. 10, we provide the basic SOQ workflow to explain the interactions between a data owner, a client, and a TEE-equipped server. (1) The data owner outsources database  $(\mathbf{d}, \Lambda)$  to the cloud server. Note that the encrypted database is stored on the untrusted domain. (2) The data owner initializes the TEE and then proves the exact program  $\Phi$  for providing SOQ services has been loaded into server's enclave successfully. (3) The client also builds a secure channel with TEE via remote attestation and then sends a query  $q$  to the cloud. (4) The cloud server employs TEE to execute the SOQ protocol honestly, which is run between the EPC and unprotected memory by `ecall` / `ocall` invocations. (5) Finally, the enclave returns the final results  $(y, \Gamma)$ , in which  $y = \Phi((\mathbf{d}, \Lambda), q)$  and  $\Gamma$  proves the program is executed honestly. Note that all messages communicated with the enclave are sent over the secure channel.

#### A.1.2 Research Status.

**Relational Database.** In [118], EnclaveDB ensures the data and the query are confidential, intact, and fresh, even if the host is malicious. The database logs are verified for achieving integrity and freshness. EnclaveDB presents a solution to implement minimal thread synchronization required for concurrent, asynchronous appending and truncation. By experiments, EnclaveDB introduces

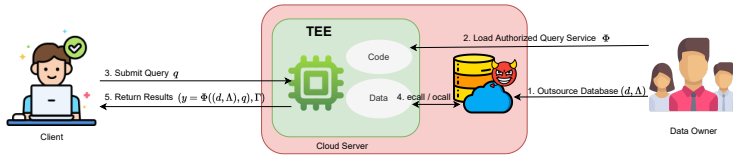


Fig. 10. Secure Outsourced Queries

about 40% overhead compared to the traditional database systems. However, EnclaveDB only works on the encryption mode, and the access pattern may leak sensitive information to the attackers.

To mitigate the side channel attacks, OblIDB was presented to provide efficient and secure SQL queries via multiple access methods [49]. For example, four oblivious SELECT algorithms are embedded in the database system, OblIDB takes advantage of knowledge about the query to choose an optimal algorithm to finish the SELECT operation. Therefore the methods apply to different settings. However, the structure of the queries is leaked to the adversary. In [39], Dang et al. presented an optimal computational model called Scramble-then-Compute (STC). STC introduced a new component (called Scrambler) to prevent leakage from access patterns, and the main idea is to permute the input data randomly before the original algorithm is invoked. In such a way, the access pattern does not leak any sensitive information while the TCB is kept lean. With STC, one can improve the efficiency of an extensive body of oblivious algorithms (e. g. sort, compaction, selection, aggregation, join). In [144], a scalable encrypted database (StealthDB) was presented to support full SQL queries, which include query services as well as database creation and transaction processing. While OblIDB achieves the strongest protection by making the user data completely inaccessible, StealthDB offers add-on features but only achieves weaker protection. In [139], Sun et al. explored the trade-offs in security, performance, and functionality. They proposed the Enclave, which leverages B+ tree to design a three-tier index (record level, buffer level, and page level). With the three-tier index, they studied the optimal node size, buffer size, and page size. Extensive experiments showed that Enclave achieves 5-9x improvements than previous schemes.

Besides, VeriDB [165] studies the verifiable general queries technologies on the relational databases via SGX. VeriDB ensures the correctness, completeness, and freshness of the query results. Note that VeriDB focuses on the verifiability of queries and is built on top of an unencrypted database. Thus it fails to provide the privacy protection of the database. To achieve verifiability, VeriDB only incurs only about 1-2 ms on read/write operations and 9%-39% overhead for the complex analytical workload.

**Key-Value Store.** To ensure confidentiality and integrity of key-value store management, a basic strategy is to load the entire key-value store to the enclave. Then a client can read or write the data via the enclave interfaces. However, the performance is very low when a large store is maintained because of the limited EPC and the page switches. ShieldStore [75] manages the key-value stores by the hash table, and it overcomes the EPC limits by maintaining an encrypted hash table in untrusted memory associated with Merkle trees for integrity protection. ShieldStore only stores the header pointer of the hash table and Merkle tree root nodes in the enclave. With the protection of enclaves, ShieldStore outperforms 8-11 times than baseline SGX key-value stores. In order to ensure freshness against rewind attacks, SPEICHER [9] borrows the idea from RocksDB by embedding an asynchronous trusted counter to an authenticated and confidentiality-preserving LSM data structure, which is maintained in the untrusted storage medium. Compared with the RocksDB, SPEICHER only incurs reasonable overheads for ensuring confidentiality, integrity, and freshness.

**Labeled Documents.** Labeled documents consist of many documents, and each is associated with one or several keyword(s). Inverted index design is very popular in labeled document retrieval

systems. An inverted index contains a set of keywords, and a document posting list of each keyword represents documents that contain the keyword. Shao et al. [129] proposed the masked inverted index (MII), which supports efficient query processing with the help of SGX.

In [52], Ferreira et al. designed the BISEN, a new Boolean Symmetric Searchable Encryption (SSE) scheme based on trusted hardware and the traditional SSE. BISEN supports multiple clients with access control features, provides verifiability against fully malicious adversaries, supports dynamic updates with forward and backward privacy, and supports arbitrarily complex boolean queries with filters. Note that BISEN only reveals which encrypted index entries are accessed. By leveraging TEEs as remote trust anchors, BISEN is able to move most client-side computations to the server, reducing computation, storage, and communication overheads. *In fact, BISEN builds the index on cloud server.* Besides, [53] and [75] showed how to build B+ tree and TF-IDF securely using Intel SGX in remote servers.

## A.2 Secure Distributed Queries

**A.2.1 Problem Statement.** Secure distributed queries (SDQ) is run between a master node and multiple slave nodes, similar as SDC described in Sec. 3.2. The data owner outsources the encrypted database to the master node and authorities the query services to the master node. The master node splits the large-scale database into small-scale databases and then distributes them to the slave nodes. When receiving a query  $q$  from a client, each slave node computes an intermediate result based on its inputs. Then the master node aggregates all intermediates into the final results. Precisely, an SDQ protocol consists of two sub-protocols  $\Psi$  and  $\Pi$ , where  $\Psi$  is run in slave nodes for obtaining the intermediate results, and  $\Pi$  is run between the master node and the slave nodes for getting the final results. Specifically,

$$\Phi((\mathbf{d}, \Lambda), q) = \Pi(\Psi((\mathbf{d}_1, \Lambda_1), q), \Psi((\mathbf{d}_2, \Lambda_2), q), \dots, \Psi((\mathbf{d}_N, \Lambda_N), q)) \quad (3)$$

Where  $(\mathbf{d}, \Lambda) \triangleq (\mathbf{d}_1, \Lambda_1) \circ \dots \circ (\mathbf{d}_N, \Lambda_N)$ <sup>2</sup>. Similar to SDC, the privacy and integrity also should be achieved in an SDQ protocol.

While many primitives to design SDQ protocols are similar to those for SOQ in an outsourced framework, SDQ protocols focus more on the compatibility with some popular distributed platforms such as MapReduce and Spark. Therefore, besides methods for achieving secure queries, we also care about whether a protocol is compatible with existing distributed platforms. As shown in Fig. 11, we provide the basic SDQ workflow to explain the interactions among a data owner, a client and the TEE-equipped nodes. (1) The data owner outsources database  $(\mathbf{d}, \Lambda)$  to master node. Note that the encrypted database is stored on the untrusted domain. (2) The data owner initializes the TEE and then proves the exact program  $\Pi$  for providing aggregation SDQ services has been loaded into master's enclave successfully. (3) The master node splits the database  $(\mathbf{d}, \Lambda)$  into small-scale databases  $(\mathbf{d}_i, \Lambda_i)$ ,  $i = 1, 2, \dots, N$ . (4) The master node also builds secure channel via remote (or local) attestation with the slaves and then ensures the distributed program  $\Psi$  is loaded into slaves' TEE successfully. (5) The client also builds a secure channel with TEE via remote attestation and then sends a query  $q$  to master node. (6) Then The master node sends  $q$  to all slave nodes, and each slave node obtains an intermediate result  $(\Psi((\mathbf{d}_i, \Lambda_i), q), \Gamma_i)$ , which is run between the enclave and untrusted host by `ecall`/`ocall` invocations. Note that  $\Gamma_i$  is employed to prove the program is run honestly in each slave node. (7) After that,  $\Pi$  is called to aggregate all intermediate results into the final results. (8) Finally, the master node returns the final results  $(y, \Gamma)$ , in which  $y = \Phi((\mathbf{d}, \Lambda), q)$  and  $\Gamma$  proves the program is executed honestly. Noted all messages communicated with enclave are sent over the secure channel.

<sup>2</sup> $\circ$  denotes the joint operation between the databases.

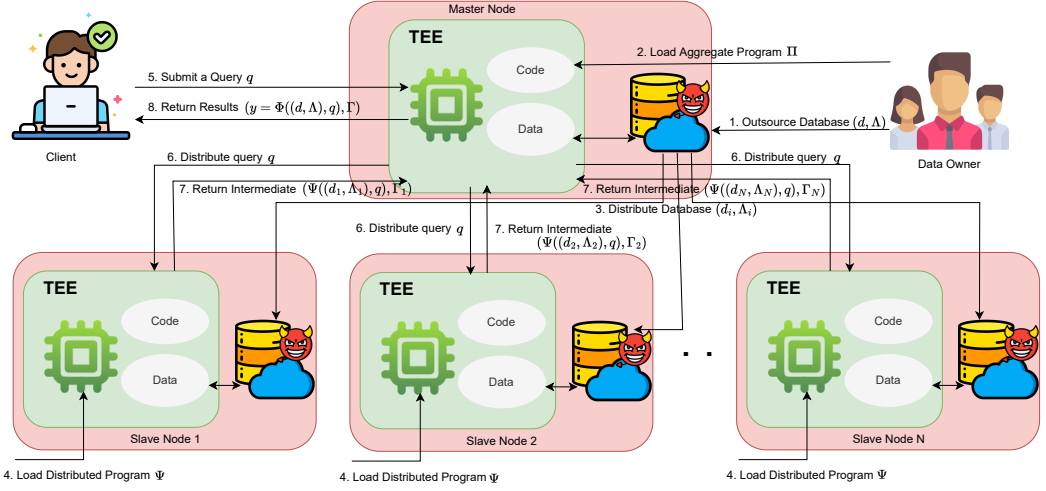


Fig. 11. Secure Distributed Queries

**A.2.2 Research Status.** In [164], Zheng et al. proposed the distributed analytics platform Opaque. Opaque promises obliviousness of SQL operations and thus achieves a strong privacy guarantee. To ensure obliviousness, Opaque introduces the Intra-machine oblivious sorting algorithm and the Inter-machine oblivious sorting algorithm, which serve as the basic building blocks for the oblivious filter, oblivious aggregation, and oblivious join. Furthermore, they also designed novel query planning techniques that improve efficiency by reducing the number of sort invocations and introducing mixed sensitive attributes. The experimental results showed that obliviousness by fully-padding mechanism comes with a 1.6-46x overhead. The fully-padding mechanism prevents information leakage from the size of intermediate results. To further improve the efficiency, Hermetic [154] applied the differential private (DP) padding mechanism to the distributed data analytics. The DP-padding method achieves 1.6-43x improvement than the fully-padding mechanism. When Hermetic only applies DP to pad the intermediate results, DPSpark [149] also applies it to the algorithms and thus introduces the notion of  $(\epsilon, \delta)$ -differentially private obliviousness  $((\epsilon, \delta)$ -DPO), which relaxes full obliviousness to enable efficiency improvements. They first presented the perturbation-shuffle-analysis framework and then designed several differentially oblivious operators (such as sort and join). The DPSpark system is optimized by reducing the number of oblivious shuffles and is benchmarked in different parameters. In [1], several classical algorithms were implemented with differential obliviousness.

For Key-value stores, Bailleu et al. proposed Avocado [8], which overcomes the physical memory limitation and thus achieves high-performance and fault-tolerance in a malicious environment. By experiments, Avocado provides a fast lookup speed of about 1.5x-9x faster than ShieldStore. For stream data, Part et al. presented the StreamBox-TZ [109], which processes large stream data in an IoT environment. StreamBox-TZ solves two major challenges: minimizes the TCB in edge devices and verifies the execution of stream analytics on edge. Moreover, on the octa-core ARMv8 platform, StreamBox-TZ exhibits performance with sub-second delay.

### A.3 Secure Multiparty Queries

**A.3.1 Problem Statement.** Like SMC, a secure multiparty queries (SMQ) protocol also targets answering a joint database query securely among several distinct yet connected parties. SMQ is

usually designed to handle complex queries. Specifically, each participant has a local database  $(d_i, \Lambda_i)$  and the goal is to design a protocol to answer a query  $q$  on the joint database  $(d, \Lambda) \triangleq (d_1, \Lambda_1) \circ \dots \circ (d_N, \Lambda_N)$ . The correctness means that the SMQ protocol completes the analytical tasks and obtains the final result correctly, and privacy means that the execution of the protocol only reveals the final result to the intended parties. Integrity ensures that all parties follow the protocol steps honestly. Finally, fairness is also defined in SMQ, which means malicious parties receive the outputs if and only if other honest parties also receive their outputs.

Many cryptographic SMC protocols can be leveraged to design SMQ protocols [13, 145]; however, they are still impractical because of either the huge communication cost or the extensive computational cost. In this survey, we focus on the hardware-assisted approaches, which improve efficiency significantly. Fig. 12 shows the workflow of a secure two-party SMQ protocol. The two parties each has a database, denoted as  $(d_1, \Lambda_1)$  and  $(d_2, \Lambda_2)$ , respectively. Then the local program  $\Phi$  is loaded into the enclave. When receiving a query  $q$ , the two parties finish the computation in an interactive manner, which consists of multiple rounds of communication. In each round, a message-proof tuple  $(m_i, \Gamma_i)$  is derived from its private inputs, and then the tuple is sent to another party.  $\Gamma_i$  is employed to prove the message  $m_i$  is computed honestly. Finally, the two parties may run a fair protocol to distribute the final result  $\Phi((d, \Lambda), q)$ , where  $(d, \Lambda) \triangleq (d_1, \Lambda_1) \circ (d_2, \Lambda_2)$  is the joint database. Noted that all messages between the two parties are sent over the secure channel.

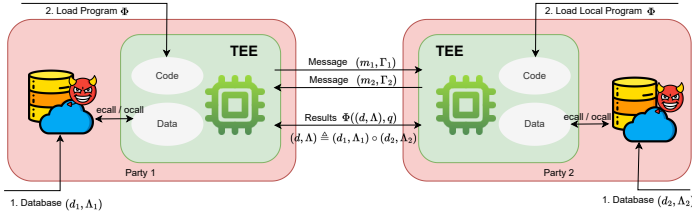


Fig. 12. Secure Multiparty Queries

**A.3.2 Research Status.** While protocols (such as SFA [121], Shrinkwrap [14], OblJoin [28, 77]) are based on differential privacy or cryptographic methods, OCQ proposed in [41] leveraged hardware enclaves to build a general framework for secure multipart analytics. Since SMQ is executed among different parties and thus it always results in high communication costs. In OCQ, parties involved in the protocol should share a schema in advance, then execute the well-designed protocols obviously for some allowed queries in the wide-area network. They also presented an optimized query planner with sensitive cardinalities, which executes the queries based on relational operators in order to prevent data leakage caused by side channels such as memory access patterns, network traffic, and other factors such as cardinality and statistics. Compared to the Opaque, OCQ is up to 9.9x faster in the wide-area network. However, the improvement is only applicable for selection and aggregation queries, and for the join queries, the cost is still not practical. The works discussed in Sec. 3.3 are workable for data analytics. However, they only work for very small-scale data sets. Although some SDQ protocols can be extended to SMQ protocols, designing practical SMQ protocols is still very challenging. Generally, it remains an open problem to design practical SMQ protocols with high throughput.

#### A.4 Comparisons

We present a comprehensive comparison of secure database queries protocols based on their setting, methodology, security features, and performance. Table 3 demonstrates that many of current

hardware-supported encrypted database systems focus on the SQL queries over relation databases, and some of them focus on set/get operations over key-value stores or keyword queries over labeled documents set. However, there are very few studies on analyzing the steam data, except for [109]. From the security perspective, keeping input/output privacy is the basic security requirement of TEE-based protocols. Employing oblivious primitives provides stronger privacy protection, for example some SOQ protocols (e.g. OblIDB [49], Enclave [49]) can avoid access pattern leakage in the memory level, whereas some SDQ (e.g. Hermetic [154], DPSpark [154]) and SMQ (e.g. OCQ [41]) protocols prevent pattern leakage from both memory and network level. While most of the existing protocols achieve integrity of IO and execution, EnclaveDB [118] and VeriDB [165] also achieve the query integrity. From performance aspects, we summarize the protocol costs with regards to different configurations and different functions. If a protocol is compared to the baseline “fully in enclave”, we employ a positive number to represent their improvements. For example, Enclave [49] improves the baseline about 5-13 times. If a protocol is compared to the baseline “plaintext model”, we employ a negative number to represent their overhead. For example, Opaque [118] introduces about -160% – -460% overhead compared to the baseline on join operations.

Table 3. Comparisons of Secure Data Queries

Schemes	Settings		Methodology		Security					Performance	
	Framework	Function	Technologies	Primitives	Security Goals			Adversarial Models	Formal proof	Configurations	Costs
					Privacy	Integrity	Freshness				
[Opaque [164], NSDI, 2017]	SDA, n	SQL	Stateful TEE + OP	AE, oSort	IO, mPattern, nPattern, IR	IO, exe	✓	Malicious Server	✗	DataSet: Relational Database Platform: Spark Number of nodes: 5 Baseline: Plaintext Model	Join: 1.6-4.6x slowdown PageRank: 3.8x slowdown
[STC [39], PETS, 2017]	SOA	SQL	Stateful TEE + OP	MS, PRP	IO, mPattern	IO, exe	✗	Semi-honest, Adaptive Server	SIM	DataSet: Relational Database Platform: Spark Baseline: Plaintext Model	Sort: -79% Select: 2.4x slowdown Aggregation: 1.2x slowdown Join: 3.8x slowdown
[EnclaveDB [118], S&P, 2018]	SOA	SQL	Stateful TEE	AEAD	IO, Query	IO, exe, Log, Query	✓	Malicious Server	✗	DataSet: Relational Database Platform: Hekaton Baseline: Plaintext Model	TATP: 0.2x slowdown
[ObliDB <sup>1</sup> [49], VLDB, 2019]	SOA	SQL	Stateful TEE + OP	AE, oSort	IO, IR, mPattern	IO, exe	✗	Malicious Server	✗	DataSet: Relational Database Platform: Spark Baseline: Opaque	Join: 1.4-2.7x speedup
[ShieldStore <sup>2</sup> [75], EuroSys, 2019]	SOA	Set/Get	Stateful TEE	AE, MAC, MHT	IO	IO, exe	✗	Malicious Server	✗	DataSet: Key-value store Platform: Ubuntu Server Number of Entries: 10M Baseline: Fully in Enclave	RD50, Z: 1.2x speedup RD95, Z: 1.6x speedup RD100, Z: 1.7x speedup
[SPEICHER [9], FAST, 2019]	SOA	Set/Get	Stateful TEE	Skiplist, LSM	IO, State	IO, exe	✓	Malicious Server	✗	DataSet: RockDB Platform: Ubuntu Server Baseline: Plaintext Model	RD90, U: 14.8x slowdown RD80, U: 15.2x slowdown RD100, U: 32x slowdown
[StealthDB [144], PETS, 2019]	SOA	SQL	Stateful TEE	AE	IO	exe	✗	Semi-honest Server	SIM	DataSet: Relational Database Baseline: Plaintext Model	0.18-1.2x slowdown
[Hermetic [154], Tech. Report, 2019]	SDA, n	SQL	Stateful TEE + DOP	DP	IO, mPattern, nPattern, IR	IO, exe	✗	Semi-honest, Adaptive Server	DP Analysis	DataSet: Relational Database Platform: Spark \$(e, \delta)\$: (5e-3, 1e-5) Baseline: Fully-Padding in Enclave	Select: 2.1-3.2x speedup Groupby: 1.8-2.7x speedup Join: 4.1-8.3x speedup
[SBT <sup>3</sup> [109], USENIX ATC, 2019]	SDA, n	Stream Algorithms	Stateless TEE	AE	IO	IO, exe	✗	Malicious Edge Devices	✗	DataSet: Taxi IDs Platform: octa core ARMv8 p Baseline: Plaintext Model	Top-k: 0.04-0.12x slowdown WinSum: 0.5-0.8x slowdown Filter: 0.5-0.9x slowdown
[BISEN <sup>4</sup> [52], TDSC, 2020]	SOA	Boolean Queries	Stateful TEE	AE + PRF	IO, Query	IO, exe	✗	Malicious Server	✗	DataSet: Labeled Document Platform: Ubuntu Server Number of documents: 5.5M Baseline: No baseline	Search: 50s Update: 2.7h
[OCQ [41], EuroSys, 2020]	SMA	SQL	Stateful TEE + OP	AE, oSort	IO, mPattern, nPattern, IR	IO, exe	✗	Malicious Server	SIM	DataSet: Relational Database Platform: Spark Number of nodes: 5. Baseline: Opaque	Aggregation: 6.7-9.9x speedup Join: 1.0-2.3x speedup
[DPSpark [149], TDSC, 2021]	SDA, n	SQL	Stateful TEE + DOP	DP	IO, mPattern, nPattern, IR	IO, exe	✗	Semi-honest, Adaptive Server	DP Analysis	DataSet: Relational Database Platform: Spark Baseline: Plaintext model	WordCount: 0.1-0.2x slowdown PageRank: 0.35-0.54x slowdown Log Query: 0.25-0.4x slowdown K-Means: 0.2-0.5x slowdown GroupBy: 0.45-0.85x slowdown
[MII [129], CIKM, 2020]	SOA	Top-k	Stateful TEE	oShuffle	IO, Query	IO, exe	✗	Semi-honest Server	Oblivious Analysis	DataSet: Relational Database Number of documents: 1.3M Baseline: [44]	Top-k: 0.29-1.5x speedup
[Enclave [139], VLDB, 2021]	SOA	Set/Get	Stateful TEE + OP	DE, B+ tree	IO, IR, mPattern	IO, exe	✗	Malicious Server	✗	DataSet: Key-value store Platform: Ubuntu Server Number of Items: 10M Baseline: Fully in Enclave	YSCB: 5-13x speedup
[VeriDB [165], SIGMOD, 2021]	SOA	SQL	Stateful TEE	MHT	✗	IO, exe, query	✓	Malicious Server	✗	DataSet: Relational Database Platform: Ubuntu Server Number of Items: 10M Baseline: No baseline	Select: 20-30s Join: 80-90s
[Avocado <sup>5</sup> [8], USENIX ATC, 2021]	SDA, n	Set/Get	Stateful TEE	BFT	IO, IR, mPattern	IO, exe	✓	Malicious Server	✗	DataSet: Key-value store Platform: Ubuntu Server Number of Items: 10M Baseline: ShieldStore	YSCB: 6-24x speedup

**AE:** Authenticated Encryption, **MS:** Melbourne Shuffle, **PRP:** Pseudo-Random Permutation, **IR:** Intermediate Results, **SIM:** Simulate-based Proof, **TATP:** The workload that consists of 40000 transactions, **MAC:** Message Authentication Code, **MHT:** Merkle Hash Tree, **LSM:** Log-Structured Merge Tree, **PRF:** Pseudo-Random Functions, **OP:** Oblivious Primitives, **DOP:** Differential Oblivious Primitives, **DE:** Delta Encryption, **oSort:** Oblivious Sort, **oShuffle:** Oblivious Shuffle, **BFT:** Byzantine Fault Tolerance;

<sup>1</sup> <https://github.com/SabaEskandarian/ObliDB.git>,

<sup>2</sup> <https://github.com/cocoppang/ShieldStore.git>,

<sup>3</sup> <http://xsel.rocks/p/streambox>,

<sup>4</sup> <https://github.com/bernymac/BISEN.git>,

<sup>5</sup> <https://github.com/mbailleu/avocado.git>;