STAP Smart Technologies Academic Press

Journal of Cyber Security and Risk Auditing

https://www.jcsra.thestap.com/

Mitigating Information Leakage Risks in Secure Multiparty Computation through Function Hiding

Udit Mamodiya[1*], Vandana Ahuja[2], Indra Kishor[3], Amer Alqutaish[4*], Rami Shehab[5] and Mansour Obeidat[6]

[1]Associate Professor, Faculty of Engineering and Technology, Poornima University, Jaipur 303905, Rajasthan, India
[2] Associate Professor, Department of CSE, MMEC, Maharishi Markandeswar Deemed to be University, Mullana, Ambala, Haryana, India
[3] Assistant Prof., Dept. of CSE, Poornima Institute of Engineering and Technology, Jaipur 302022, Rajasthan, India
[4]Deanship of Development and Quality Assurance, King Faisal University, 31982, Al-Ahsa, Saudi Arabia
[5]Vice-Presidency for Postgraduate Studies and Scientific Research, King Faisal University, 31982, Al-Ahsa, Saudi Arabia
[6]Applied College, King Faisal University, Al-Ahsa, Saudi Arabia

ARTICLE INFO

Article History
Received: **07-12-2025**
Revised: **17-01-2026**
Accepted: **25-01-2026**
Published: **26-01-2026**

Vol.2026, No.1

DOI:

Corresponding author. Email:
assoc.dean_research @poornima.edu.in
and
aalqutish@kfu.edu.sa

Orcid:

https://orcid.org/0000-0002-8022-4252

ABSTRACT

Secure multiparty computation (SMPC) allows a joint computation on private data, but the majority of the existing protocols assume implicitly that the computation being performed is public and non-sensitive. In practice, though, computation logic frequently incorporates proprietary strategy or sensitive rule of decision, and its exposures constitute a significant though neglected leak of information. The prevailing SMPC models can mainly provide input confidentiality and accuracy, but the protocol level leakage due to observable protocol behavior is not well tackled. This paper attempts to fill this gap by introducing a (Function-Hiding Secure Multiparty Computation) FH-SMPC framework modifying the SMPC workflow to incorporate encapsulation of functions and regular patterns of execution directly. The suggested design hides structural and semantic attributes of the considered function and maintains the correctness, scalability, and compatibility with the conventional SMPC primitives. An explicit security analysis provides indistinguishability of functions along with classical input privacy. Experimental evaluation shows that FH-SMPC reduces transcript-based function distinguishability by over 85%, achieving an average divergence of 0.028 compared to 0.214 for baseline SMPC, with an execution latency increase limited to approximately 12% and no asymptotic growth in communication overhead. By treating computation logic as a confidential asset, FH-SMPC advances secure collaborative computation and provides a practical foundation for privacy-sensitive applications in cloud analytics and distributed decision systems.

Keywords: Secure Multiparty Computation, Function Hiding, Information Leakage, Cryptographic Protocols, Privacy-Preserving Computation.

How to cite the article

1. Introduction

Secure Multiparty computation (SMPC) has become a paradigm of cryptography to allow collaborative computation of parties acting in mutual distrust without having to reveal their own inputs. Since its formalization in the late twentieth century, SMPC has developed as a theoretical framework to a practical instrument in the basis of privacy supporting data analytics, distributed machine learning, federated optimization, and secure decision-making across organizational borders [1]. The basic idea behind SMPC is that it allows calculating any arbitrary functions on private data, and ensures that no party learns anything except as can be deduced by their own contribution and the resulting output. This attribute has made SMPC to be essential in fields where regulatory authorities, market competitiveness or ethical factors forbid sharing of raw data.

The security discourse has however evolved beyond classical input privacy as SMPC protocols have evolved and now been applied to a real world scenario. Modern usages are more and more showing that it is not enough to protect inputs. In most collaborative environments, the very form of the computation or decision logic or analytical model under consideration is highly sensitive intellectual property. As an example, proprietary risk-scoring models, strategic allocation functions and private analytics pipelines may enshrine domain expertise that the participants are either unwilling or incapable of sharing. The conventional SMPC models though effective in input protection, often assume that the role being calculated is social and is entirely known by all members [2]. The reason behind this assumption being reasonable and reasonable in the abstract world of academics however turns out to be a problem in the high-stakes and practical world.

The revelation of functional information presents a low but significant attack surface. Although the inputs are encrypted and intermediate values masked, the fact that the exact implementation of the function is known can be exploited by the adversaries to tell sensitive relationships, reverse-engineer the proprietary logic, or correlate the outputs with previously known structural properties of the computation [3]. Such leakage can be not considered a breach of the formal notion of SMPC security in an adversarial or semi-honest environment, but can still lead to economic or strategic damage. Function-level confidentiality has therefore become one of the most important aspects of privacy-preserving computation, albeit having gone under-discussed.

Function hiding as the concept attempts to overcome this risk by assuring that the parties involved in the initiative do not know the inner workings or the semantic information about the evaluated function. Rather, parties engage with an abstract able or obfuscated representation which maintains a model and masks functional purpose. Initial discussions of this concept have been made in the sense of the evaluation of private functions (PFE) where one party has a private function and the other parties have private inputs [4]. PFE protocols prove that it is possible to effectively hide functions, but they are usually based on special constructions, incur a lot of computational cost, or make a restrictive assumption about the adversarial model. In addition, most approaches to PFE are restricted to Boolean circuits or particular functions classes, restricting their use to general-purpose SMPC.

In line with these innovations, there is also the topic of functional encryption (FE) as a potent cryptographic primitive that can disclose only output functions out of encrypted input [5]. FE, by construction, enables a decrypt or to be able to learn f(x) but nothing more about xxx, when fff is embedded in the decryption key. This treatment of information and functionality asymmetrically indicates one interesting potentiality: is it possible to have function hiding by storing functional logic in cryptographic objects instead of putting it on the protocol level? Although FE in itself is theoretically elegant, it fails to consider the collaborative, multi-party aspect of SMPC, and does not necessarily hide the structure of functions among all participants in a distributed environment.

A second approach that has been explored is program obfuscation as a method of concealing the logic of computation and still remaining executable [6]. More precisely, indistinguishability obfuscation (iO) provides good theoretical guarantees, and ensures that, obfuscated programs do not provide significantly more information than black-box access. But, iO constructions are still computationally intensive and based on complicated assumptions of hardness and not yet feasible to large scale applications or applications which are sensitive to latency. Moreover, the direct encoding of obfuscated programs into the SMPC workflows creates difficulties with regards to the composability, verification, and predictability of performance.

Consequently, current methods of the function hiding are more likely to examine individual facets of the issue but not provide a unified approach to it. Others focus on high levels of theoretical guarantees rather than high levels of practicality and others focus on efficiency but use constraining assumptions or limited threat models [7]. Function disclosure in the majority of deployed SMPC systems is still implicitly admitted as a necessary trade-off. This lack of theoretical potential and real implementation points to the necessity of a systematic reconsideration of the way the issue of the function confidentiality is addressed in the context of the SMPC protocols.

The issue is even worsened by the increased complexity of the contemporary collaborative computations. In contrast to the early SMPC application scenarios of simple arithmetic or voting protocols, modern applications are usually characterized by stratified decision making, adaptive control flow and data-related branching. In this case, divulging the role may unwillingly divulge the strategic priorities, optimization criteria or acquired model behaviors [8]. Although the inference attacks might be mitigated with additional information, even the partial information about the computation graph can be used to carry out the inference in an undesirable way.

The overwhelming opinion of most SMPC security analysis remains on input privacy, correctness and opposition to malicious deviation. The leakage of the functions is either assumed or not to affect the core security definition [9]. This dismissal implicitly will leave a blind spot in both the theoretical models and practical implementations. To overcome this blind spot, a patch will not do but a conceptual expansion of the SMPC security goals to make the hiding of functions a first-class requirement.

This paper claims that the issue of information leakage in SMPC requires a consistent structure, which combines the concept of function hiding with the concept of existing multiparty computation. Instead of assuming function confidentiality as an external augmentation, we suggest introducing it at the core of the computation process by a hybrid cryptographic design. The main concept is to separate the functional semantics and the execution of the protocols so that participants can collectively compute the results, without any participant being aware of the underlying logic that was used to derive these results.

Towards this end we present a functional hiding of SMPC framework based on lightweight functional encryption with the abstraction of protocol-level structure. The approach proposed does not need one party which owns a function and restrictive circuit representations as it would be in conventional PFE schemes. Instead, it facilitates joint analysis in the semi-honest and malicious adversarial settings, as well as being compatible with the usual constructs of SMPC building blocks including secret sharing and secure reconstruction [10]. Through a well-coordinated representation and invocation of functional components, the framework reduces leakage at a cost that is prohibitive to computation.

One of the design principles that will be applied to the proposed solution is composability. SMPC deployments in the modern world are rarely deployed in isolation; they exist as part of larger systems which include authentication, access control, auditing and adaptive decision-making. Any functional hiding of any functionality will therefore have to coexist with the existing protocols and security proofs. The suggested architecture is designed in such a manner that it can be easily extended or scalable with function hiding that can be superimposed on the existing SMPC implementations [11].

Theoretically, the framework builds on the traditional definition of SMPC security by providing the concept of function indistinguishability as a property. Informally, this guarantees that an adversary, on learning a protocol transcript, intermediate messages as well as output, will not be able to differentiate between executions of different functions, so long as the functions yield the same output on the specified inputs [12]. This concept is intuitively consistent with security models that are simulated and it is possible to reason formally about leakage boundaries.

This work is not only novel in its use of cryptographic construction, but it is also novel in how the operation of hiding functions is implemented within SMPC. Instead of extensive obfuscation or completely homomorphic assessment, the new scheme helps use selective disclosure by use of cryptographic encapsulation. This design is a carefully chosen compromise between the strength of security and computational feasibility, so this technique is applicable to a real-world implementation setting [13].

1.1 Contributions and Novelty

Unlike the previous works that present function privacy in the limited scope, the paper provides a cohesive treatment that cuts across threat modeling, protocol and leakage analysis. The framework is measured with respect to representative attack vectors and it has been shown that the use of function hiding can mitigate inference risk whilst adversaries have auxiliary information or are colluding inter-phase protocols [14]. Significantly, the analysis shows that one of the situations in which traditional SMPC would be formally safe but practically weak by the nature of its design are in need of the proposed extension.

Overall, the following contributions are made in this work. To start with, it defines the problem of leakage in functions as a serious and under-researched weakness of current SMPC systems. Second, it suggests a hybrid framework of functions hiding, which glues well with traditional primitives of SMPC. Third, it makes indistinguishability of functions in a simulation-based security model formal and offers a rigorous basis of analysis. Lastly, it shows that the hiding of strong functionality is possible without compromising practicability, thus putting the gap between cryptographic theory and secure computation in the real world closer [15].

Three main contributions are made in this work. Firstly, it singles out as a specific and practically applicable risk of SMPC the concept of leakage at the functional level, in addition to the classical emphasis of input and output confidentiality. Second, it proposes the FH-SMPC framework that implements the encapsulation of functions and repeatable execution patterns in the SMPC protocol flow to provide the indistinguishability of functions without compromising on correctness or scalability. Third, it offers formal security analysis and empirical validation that indicate that the proposed design substantially inhibits transcript based inference and remains practical.

1.2 Positioning with Respect to Prior Work.

The suggested FH-SMPC framework is fundamentally different than the current methods of using predicate functional encryption (PFE) based, functional encryption (FE) based, and program obfuscation based methods. PFE and FE are mainly aimed at centralized cryptographic-based environments, where a single evaluator views a limited or secret output of a function yet they do not consider function privacy in interactive multiparty computation. FH-SMPC, on the other hand, interoperates with the SMPC protocol flow, so that jointly computing an obscured function by multiple parties can be performed without each party having to know anything about its structural or semantic characteristics. Likewise, although obfuscation methods are designed to convert programs into forms that are meaningless to the computer, they are generally applied to individual code and not distributed-execution transcript-based leakage. The innovation of FH-SMPC is thus in the indistinguishability of its functionality at the protocol level, as opposed to the use of individual cryptographic primitives, making computation logic protection in collaborative and distributed systems practicable.

1.3 Research Objectives.

This study will be motivated by the following research questions because the current SMPC protocols exhibit certain limitations which do not adequately combat the logic of computation:

1. to define formally the function level leakage due to the visible protocol traces and interaction patterns in the multi-party computation;
2. to come up with an FH-SMPC architecture that introduces the concept of function hiding directly into the SMPC protocol architecture without sacrificing any correctness or privacy of the input;
3. to achieve security assurances of both semi-honest and malicious adversarial models of function indistinguishability; and
4. to empirically test the efficacy of the suggested solution to decreasing transcript distinguishability and adversarial benefit in contrast to baseline SMPC schemes.

This paper will help improve the state of privacy-preserving computation by incorporating function confidentiality as an aspect of SMPC security. The rest of this paper is organized in the following way. Section 2 operates a literature review and critical analysis of the available literature on secure multiparty computation, leakage-resilient cryptographic primitives, and function-hiding techniques in order to identify the conditional limitations that underlie this research. Section 3 outlines the suggested methodology, which involves the threat model, design principles and the function-hiding SMPC framework that has been devised to address the risks of information leakage. Section 4 introduces the experimental setting and findings, which indicate the effectiveness and efficiency of the given approach. Section 5 presents the results in the context of the already existing work and addresses the issue of security implications, practical concerns, and possible trade-offs. Lastly, Section 6 wraps up the paper and gives future research directions such as extending to more powerful adversarial models and real-life implementation conditions.

2. Literature Review

2.1 Foundations of Secure Multiparty Computation

The study of secure multiparty computation has been growing at a rapid pace in the past twenty years due to the increasing need to perform joint analytics without jeopardizing the data confidentiality. Initial foundational literature proved the possibility of collective computation of functions on private inputs with strong cryptographic guarantees where much attention was paid to correctness and privacy of inputs when adversaries acted semi-honestly and maliciously [16]. These experiments formed the basis of current SMPC procedures but mostly considered that the role under test was social knowledge. This assumption seemed plausible at the time, because the majority of target applications incurred consensual and transparent calculations.

With SMPC approaching a closer to a practical deployment, research on efficiency and scalability issues commenced later. Secret sharing based and garbled circuit based protocols and homomorphic encryption protocol were also optimized to minimize the communication rounds, cost of computation, and latency [17]. Although these advances made a significant improvement, they did not change the security model radically. The function itself was still explicit, described right in protocol descriptions or circuit descriptions. Therefore, functional disclosure was viewed as a reasonable trade-off, and not as a possible weakness.

2.2 Function Privacy: PFE, FE, and Obfuscation

In parallel with SMPC evolution, the cryptography community studied the problem of private function evaluation and made it a separate problem. PFE studies considered situations when one of the parties possesses a clerical role and other parties deliver inputs, and the aim is to avoid the disclosure of either of the parts [18]. These methods showed that theoretically function concealment was possible, but the methods could be highly complicated circuit transformations, or ad hoc assumptions. Besides, the majority of PFE schemes were either less expressive or had overheads that made them impractical to use in large multiparty environments.

Another view of selective disclosure of information was brought about by functional encryption. With FE schemes, the decryption keys are associated with particular functions and only authorized computations can be done with encrypted data [19]. This paradigm provided access control at the fine-grain and also generated the interest in the privacy preserving analytics. Nonetheless, FE is concerned more with data confidentiality and not the joint calculation between two or more parties. Incorporating FE into SMPC poses nontrivial threats, such as important management, and also trust, and consistency with distributed execution models [20]. Due to this reason, FE-based solutions have been kept very abstract regarding the full-fledged multiparty protocols.

Indistinguishability obfuscation and program obfuscation have also been suggested as a style of hiding computation logic [21]. Theoretically, obfuscated programs are as revealing of information as black-box access to the functionality. There were a number of studies that indicated that obfuscation would secure proprietary algorithms in a distributed setting. However, it is still restricted by expensive computational models, strong hardness assumptions and challenges in constructing obfuscated programs to interact with interactive protocols [22]. Such constraints have confined the use of obfuscation to limited or experimental use of the practice of hiding functions.

2.3 Leakage via Execution Traces and Protocol Structure

Recent literature has come to admit that disclosing function structure may give way to indirect information leakage, despite input privacy being established on paper. Scientists have demonstrated that opponents can use information about the computation graph, branching logic or parameter dependencies to deduce sensitive properties or strategic purpose [23]. These attacks tend to be beyond the mainstream SMPC threat models and the inconsistency between formal definition and actual risk. Systematic solutions are, however, scanty in spite of this realization.

Other works have tried to obscure the details of functions by randomizing the circuit or abstraction to protocol level. The goals of these methods are to render observed executions less informative by additive noises or variability to the computation flow [24]. Although these methods may be used to minimize leakage in certain situations, they lack powerful, demonstrable properties of function indistinguishability. In a great number of cases, even repeated executions or supporting information can allow enemies still to obtain valuable information about the underlying reasoning.

2.4 Hardware- and Leakage-Resilient Approaches

SMPC has also been studied together with trusted hardware environments like secure enclaves [25]. These systems seek to protect computation logic by performing sensitive functions within hardware-secured areas to prevent participants. Nonetheless, imposing trust in hardware means creating additional vulnerabilities to attack, such as side-channel attacks and dependency on vendors. Moreover, these solutions might not be satisfactory in environments where it is not always possible to create hardware trust.

A different line of research has concentrated on leakage-resilient computation, and focused on restricting what can be learned about the protocol transcripts and side channels [26]. These works present very useful information on how to limit information exposure, but most of them focus on data leakage and not on the confidentiality of functions. When referred to, function hiding is usually only seen as a nice-to-have feature, not as a necessity.

Comparative views on SMPC frameworks indicate that a majority of the present systems focus on efficiency, scalability and resiliency to active adversaries, implicitly believing in the benign nature of disclosed functions [27]. Even sophisticated protocols that are made to address malicious security will seldom address the role as a secured resource. Such omission is more problematic in a competitive or a regulated context, where the business logic or decision policies are sensitive in the form of the encoded functionality.

The gap is starting to be pointed out in recent surveys on privacy-preserving computation. According to authors, the further SMPC systems should consider the larger concepts of privacy, such as protection of models, algorithms, and the intent of analysis [28]. These discussions, however, do not go in any further to suggest any concrete and implementable mechanisms. The literature is therefore the increasing awareness in the absence of converging methodological approaches.

There is limited effort in trying to formalize function privacy in SMPC security definitions. Some theoretical literature has offered extensions to simulation based security that add function indistinguishability [29]. Although these models are sometimes promising, they are seldom realized in actual protocols creating a disjuncture between theory and practice. The only way to fill this gap is to have designs that can be shown to be provably secure and operationally feasible.

2.5 Privacy-Preserving Biometric and IoT Computation

Recent developments in privacy-preserving biometric systems further illustrate the importance of concealing not only data but also computational intent and access patterns. In particular, privacy-enhanced facial recognition schemes based on homomorphic encryption enable encrypted-domain feature extraction and matching for IoT environments, thereby preventing direct disclosure of biometric templates during cloud-based processing. Similarly, hidden facial verification frameworks combining homomorphic encryption with privacy information retrieval (PIR) aim to prevent servers from learning which identity or feature index is being queried during verification. Although these approaches are not formulated within the SMPC paradigm, they demonstrate that practical privacy threats often arise from observable computation and query behavior rather than raw data leakage alone. These systems therefore provide empirical support for the relevance of function-level privacy in distributed and cloud-assisted analytics, reinforcing the motivation for function-hiding mechanisms in collaborative computation settings.

2.6 Summary of Research Gaps

Overall, the available literature shows that there has been significant improvement on the task of securing the inputs and outputs in multiparty computation yet presents a obvious weakness in the effort to handle the issue of information leakage on the function level. PFE, functional encryption, obfuscation, and hardware-based isolation are all approaches that provide a partial solution but do not provide a unified and efficient and generally applicable framework of hiding functions in SMPC [30]–[35]. This disjointed landscape highlights why there is a need to have in place integrated solutions that present the concept of functional confidentiality as a first-class security goal and not an exception or an afterthought. The current work continues based on these insights and suggests a single framework, which incorporates the concept of function hiding that is directly implemented into the SMPC protocol structure. It attempts to bridge the gap found in previous research by combining concepts of functional encryption and protocol abstraction, and finding a feasible route to the objective of leakage-resilient collaborative computation. Solutions based on PFE, functional encryption, obfuscation and hardware-assisted isolation are all partial solutions, and each one of them lacks a consistent and efficient framework to conceal computation logic in interactive SMPC protocols. Besides, recent privacy-conscious biometric and IoT systems demonstrate that hiding computational intent and access patterns is usually as important as data encryption. Although the importance of these risks has been increasing, the existing SMPC designs do not consider the considered functionality as a secret asset. Such a fragmented topography indicates that a combined set of protocol-level solutions is required with the express aim of achieving function indistinguishability and classical input privacy. The current contribution is based on these lessons and introduces a combined FH-SMPC architecture that directly incorporates the notion of function hiding into the SMPC protocol architecture, thus closing the gulf between the theoretical concepts of privacy of functions and actual collaborative computation.

Table 1. Comparative literature gap analysis highlighting limitations of existing secure multiparty computation and function-hiding approaches

S. No.	**Reference No.**	**Title / Focus Area**	**Methodology / Tools Used**	**Key Findings**	**Limitations / Gaps Identified**	**Relevance to the Current Study**
1	S. R. H. Najarkolaei et al. / 2024 / [1]	Beyond Yao's Millionaires: Secure Multi-Party Computation of Non-Polynomial Functions / Extending SMPC expressiveness	Protocol design for evaluating non-polynomial functions; secure computation formalization; efficiency-aware constructions	Demonstrates that SMPC can move beyond standard polynomial/arithmetic assumptions and still remain secure and correct for richer function classes	Focus remains on correctness and feasibility; function visibility is not treated as a leakage vector, so proprietary logic can still be exposed through the public function description	Helps motivate that modern SMPC supports complex functions—precisely the setting where function disclosure becomes costly, strengthening the need for function hiding in our work
2	C. Hazay et al. / 2024 / [7]	Protecting Distributed Primitives Against Leakage: Equivocal Secret Sharing and More / Leakage resilience in distributed primitives	Leakage-resilient secret sharing; equivocation techniques; formal leakage models and security arguments	Provides rigorous methods to harden distributed primitives against side leakage, supporting more robust composability	Primarily addresses leakage from shares/primitive exposure; does not directly model "function leakage" (i.e., leakage of what is being computed) as a first-class goal	Supplies provides conceptual and technical foundations on leakage-conscious SMPC building blocks, which we expand to the level of functional discretion
3	R. R. Cohen et al. / (book chapter) / [9]	SMPC with Identifiable Abort via Vindicating Release / Accountability and fairness in SMPC	Protocols that detect/attribute aborts; controlled release mechanisms; stronger execution integrity	Shows how SMPC can incorporate accountability and controlled abort handling, reducing denial-style disruptions	The focus is abort attribution, not secrecy of computation logic; participants still learn the function, and transcript structure may still reveal logic	Relevant in the sense that function hiding should coexist with realistic SMPC concerns (abort/fairness); our work focuses on the dangers of function leakage and is consistent with the existence of sound execution models
4	Q. Ye and B. Delaware / 2024 / [16]	Taypsi: Static Enforcement of Privacy Policies for Policy-Agnostic Oblivious	Static analysis + oblivious computation enforcement; program-level privacy constraints;	Treats privacy as a programmable constraint and enforces obliviousness properties at the	Emphasizes policy enforcement and obliviousness, yet function confidentiality is not the core objective; the	Closely spiritually impregnated: it demonstrates that privacy can be implemented in the programming layer--our paper equally insists that the hiding of functions should

		Computation / Policy-driven oblivious computation	compilation/verification style tooling	computation level	policy/structure may still hint at computational intent	be engineered in to SMPC rather than placed on a case-by-case basis.
5	Y. Li et al. / 2025 / [20]	Function-Hiding Multi-Client Inner-Product Functional Encryption (No Pairings) / Function-hiding FE	Pairing-free FE constructions; multi-client setting; large-space support; formal indistinguishability notions	Achieves function-hiding FE under efficient design choices, suitable for large domains and multi-client environments	It is FE-focused, not a full SMPC protocol; does not address interactive multiparty transcripts, collusion patterns, or SMPC reconstruction leakage	Extremely central: offers tangible cryptographic path to conceal function semantics, which our SMPC framework can use to minimize information disclosure concerning logic implemented
6	V. R. Cadambe et al. / 2023 / [22]	Differentially Private Secure Multiplication: Hiding Information in the Rubble of Noise / Noise-based hiding in secure arithmetic	Differential privacy integrated with secure multiplication; information-hiding via controlled noise; theoretical bounds	Illustrates that privacy can be strengthened by mixing cryptographic security with statistical hiding	Noise can distort utility; it targets value leakage more than function secrecy; does not guarantee function indistinguishability in SMPC transcripts	Our work explains the reason why, in addition to randomized outputs, structural secrecy is required by the function hiding
7	Y. Luo et al. / 2024 / [28]	Cloud-SMPC: Two-Round SMPC based on LWE / Practical SMPC efficiency and assumptions	Two-round SMPC design; LWE-based security; cloud-oriented deployment viewpoint	Shows modern SMPC can be made round-efficient and suitable for cloud settings	Efficiency-centric; function is assumed public, and the protocol's structure may leak computation intent even when inputs are protected	Closely connected to the deployment: the hiding of functions should not ruin usefulness. This paper inspires our intention to introduce layers of functionality without violating the performance requirement
8	Y. Bai et al. / 2024 / [30]	Secure Network Function Computation: Function-Security / Function-security as a formal goal	Defines and studies "function-security" in network computation; theoretical security definitions and bounds	Explicitly elevates the function as a protected object, offering a formal lens for function confidentiality	Primarily in network function computation setting; translation to general SMPC workflows and protocol transcripts needs additional design work	One of the most conceptually parallel sources: is in agreement with our basic thesis that the hiding of functions must be formalized, and then realized in SMPC by leakage conscious mechanisms

Table 1 shows a narrow gap analysis of selected and closely related works in the area of secure multiparty computation, leakage-resilient cryptography primitives, and function-hiding. Although earlier literature reveals considerable progress in the expressiveness of SMPC, efficiency, and resistance to adversarial actions [1], [7], [9], [28], the majority of the frameworks still assume that the evaluated function is publicly accessible, thereby not considering the leakage of functions at the information level. Similar work on functional encryption and function-security models develops powerful theoretical frameworks of concealing computation logic [20], [30], but such solutions are not often incorporated in interactive SMPC protocols. This analysis has identified a gap in the research: lack of a single, practical SMPC framework, which explicitly reduces information leakage due to the disclosure of functions, but does not compromise protocol efficiency and composability, which is specifically handled in the current study.

3. Methodology

In this section, the researcher outlines the proposed methodological framework that can be used to reduce information leakage in the context of secure multiparty computation as a result of information disclosure during functions. The methodology has been developed following two principles, namely, to go beyond standard SMPC security assurances of input privacy to provide explicit confidentiality at the level of functionality; and to make the resultant framework practical, modular and reproducible. The section starts with the description of the system design goals and slowly develops the threat model, problem formulation, and algorithmic foundations, which are the basis of the presented approach of function-hiding SMPC.

3.1 System Overview and Design Objectives

The suggested methodology presents Function-Hiding Secure Multiparty Computation (FH-SMPC) scheme that allows joint computation and at the same time safeguard both private inputs and privacy of computation protocol itself. Unlike the traditional SMPC models where the value of the evaluated function is presupposed publicly known, the FH- SMPC model considers the function as a valuable asset, the disclosure of which can result in indirect inference, strategic leakage, or revealing of proprietary decision-making.

On a high level, the system is a set of several distrustful parties that possess their own input data, which are private and obscure, and who strive to compute some target function collectively without exposing their inputs or the functional form in which the computation is to be done. As depicted in Figure 1, the framework proposes a special purpose, encapsulation layer, which abstracts and cryptographically shields the function before secure evaluation. This encapsulation guarantees that parties dealing with it are only involved in interchange with an encoded version of the functionality, and not with its logical form.

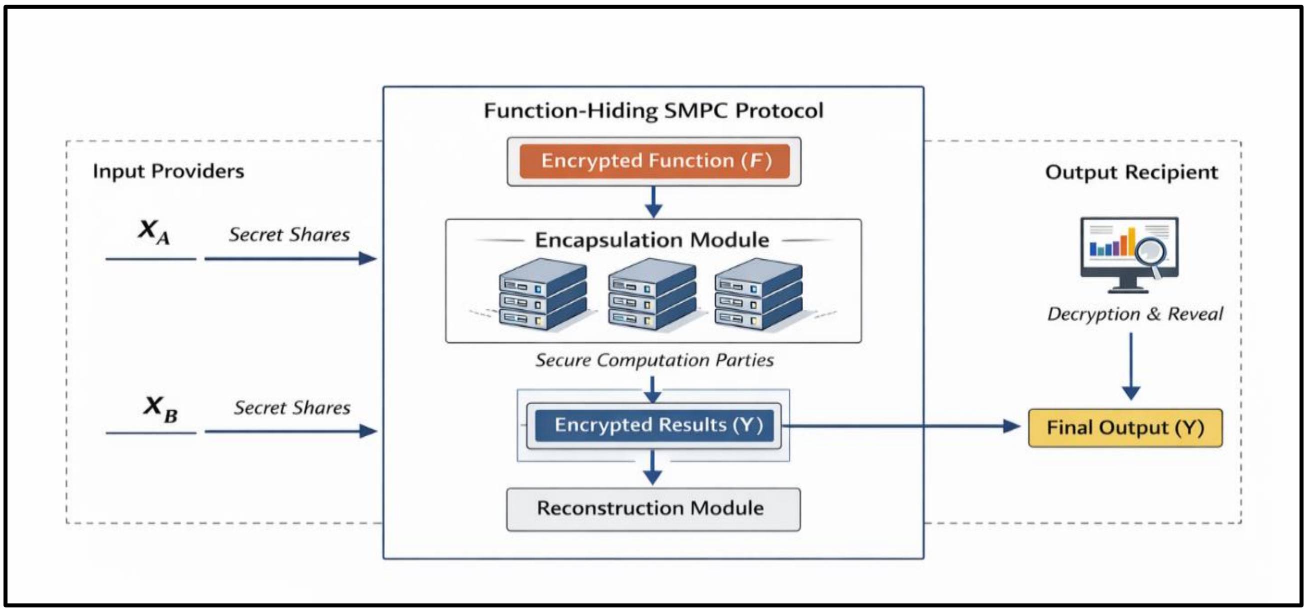

Figure 1. High-level architecture of the proposed function-hiding secure multiparty computation (FH-SMPC) framework.

In Figure 1, the architectural movement shows the distinction between semantics of functionality and semantics of execution. The preparation phase is performed first on the function owner (or agreed-upon computation logic) whereby the encapsulation of the function is done by using cryptographic abstractions. This secure representation is then used by standard SMPC components, which include sharing of secure inputs, distributed evaluation and reconstruction of the results. During the execution of protocols, middle-level messages or visible transcripts do not reveal any significant information regarding the internal structure of the functionality.

The FH-SMPC framework design has three design objectives. One, it maintains the classical properties of SMPC (such as correctness and input privacy) with semi-honest and malicious adversarial models. Second, it formally introduces the concept of indistinguishability of functions as the security goal, that is, the fact that adversaries cannot tell between two different functions that generate the same output given the same input. Third, the framework focuses on deployability as it is compatible with common SMPC primitives and does not rely on any unrealistic assumptions or heavy cryptographic schemes.

In the semi-honest adversarial model, corrupted parties are assumed to follow the prescribed protocol steps correctly but may attempt to infer information about the evaluated function from their local views and observed transcripts. For example, a semi-honest party may analyze the sequence of messages, intermediate share distributions, or execution timing patterns to distinguish whether the computation corresponds to a threshold function, a polynomial evaluation, or a conditional decision rule. Such attacks rely on passive observation and statistical inference rather than protocol deviation. In contrast, a malicious adversary is allowed to deviate arbitrarily from the protocol in order to amplify function-level leakage or disrupt the intended hiding mechanism. Examples of representative attacks in this context are injecting bad shares to cause alternative lines of execution and selective aborting the protocol to correlate termination behavior with shape of functions, and manipulation of interaction patterns to cause distinguishable transcript signatures. These active strategies aim to exploit structural dependencies between protocol behavior and the concealed function. The FH-SMPC framework is designed to mitigate both passive inference by semi-honest adversaries and adaptive manipulation by malicious parties through function encapsulation and uniform execution patterns.

3.2 Threat Model and Adversarial Assumptions

The security of the proposed Function-Hiding Secure Multiparty Computation (FH-SMPC) framework is examined in the light of a transparent and broad threat model that includes a traditional input leakage as well as a functional leakage of information. Contrasting the classical SMPC formulations that assume that the evaluated function is of a public and non-sensitive nature, this work considers the actual value of the function as secret and may be exploitative in case it is revealed during the execution of protocols. In the proposed FH-SMPC framework, function encapsulation is performed in a distributed manner rather than by a single trusted party. Each participant locally applies the encapsulation transformation to its designated share of the function representation using the function security compiler and randomization module. As a result, no individual party obtains a complete or explicit description of the encapsulated function f~(·). This distributed encapsulation process prevents concentration of knowledge about the computation logic at any single point and aligns with the threat model in which all parties are considered potentially curious or adversarial.

3.2.1 System Setting and Adversary Scope

Suppose there are n mutually distrustful parties denoted by, P=P1,P2,…,Pn and each party Pf has an input which is non-publicly known, xiномXi. The strategic goal of the parties is to collectively calculate a function on their shared inputs and maintain input privacy and function privacy. The target computer is defined like this (1).

$$y = f(x_1, x_2, \dots, x_n) \cdots (1)$$

In which f(.) is the function that needs to be jointly evaluated by the participating parties, and y represents the common output of the computation.

In this case, xi represents the individual input of party Pi and Xi represents the domain of its input. The opponent A is thought to be probabilistic poly-time (PPT), having complete access to protocol messages and the capacity to corrupt a group of the participating parties. It does not make any assumptions about a trusted third party at any point during the protocol.

3.2.2 Adversarial Behavior Models

Two normative adversarial behaviour are taken:

1 .Semi-honest adversaries, which adhere to the protocol specification but seek to extract extra information by all information they observe when running.

ISSN: 3079-5354

2. Malicious attackers, who can arbitrarily noncompliantly follow the protocol, inject corrupted messages, adaptively decide inputs, or seek to control protocol execution to steal unauthorized information.

In both cases, the adversary may collude with up to t<n parties and aggregate their local views, including randomness and message histories.

3.2.3 Observable Transcripts and Leakage Channels

Let T denote the observable execution transcript of the protocol, defined as (2).

$$T = \{m_1, m_2, \dots, m_k\} \cdots (2)$$

Where each m_j represents a protocol message or observable execution artifact, including message ordering and communication metadata. While classical SMPC guarantees ensure that T does not reveal private inputs beyond what can be inferred from the output y, this assumption does not hold when the function f itself encodes sensitive logic. In such cases, transcript structure and interaction patterns may leak information about the function semantics.

3.2.4 Function Leakage and Indistinguishability Criterion

To formalize function-level leakage, consider two distinct functions f0 and f1, defined over the same input domain, such that (3).

$$f_0(x_1, \dots, x_n) = f_1(x_1, \dots, x_n) \forall (x_1, \dots, x_n) \in X \cdots (3)$$

An FH-SMPC protocol is said to satisfy *function indistinguishability* if no adversary can determine whether the protocol was executed with f0 or f1 based solely on the observed transcript. Formally (4),

$$\boldsymbol{Pr}\left[A\left(T_{f0}\right) = 1\right] - \boldsymbol{Pr}\left[A\left(T_{f1}\right) = 1\right] \leq \varepsilon(\lambda) \cdots (4)$$

where ε(λ) is a negligible function in the security parameter λ, and Tfi denotes the transcript generated during evaluation of function fi. Equation (4) captures the core security requirement of FH-SMPC: transcripts must not leak distinguishable information about the underlying function.

3.2.5 Collusion and Adaptive Inference

The attacker can dominate a colluding group of parties' C ⊆ P, 0 C -t, and combine all available views to increase the inference capacity. Adaptive attacks: These are attacks in which inputs are selected according to previous output or transcript observations. FH-SMPC framework applies the same interaction patterns to executions to avoid such adaptive correlation attacks.

3.2.6 Threat Model Illustration

Figure 2 depicts the general threat landscape that was taken into account in this paper. Figure 2 highlights the interaction between honest parties and adversaries, emphasizing how function encapsulation restricts information flow even in the presence of collusion and transcript visibility.

3.2.7 Security Scope

The threat model presupposes the standard assumptions of cryptographic hardness and secure channels of communication. We do not consider hardware-level side channels or denial-of-service attack as this work. Under these premises, the FH-SMPC model gives effective assurances against leakage of inputs and disclosure of functions on an information level.

3.3 Problem Formulation and Function Leakage Quantification

This sub section formalizes the issue of information leakage at the functional level in the context of a secure multiparty computation and forms the basis of analysis of the proposed FH-SMPC architecture. Although classical SMPC models are aimed at deterring unauthorized exposure of personal inputs, they implicitly presuppose that the considered functional is non-sensitive and public. In most realistic collaborative environments, though, the actual functionality contains confidential decision logic, optimization, or proprietary analytics, and the revelation of this functionality is a critical privacy and security threat.

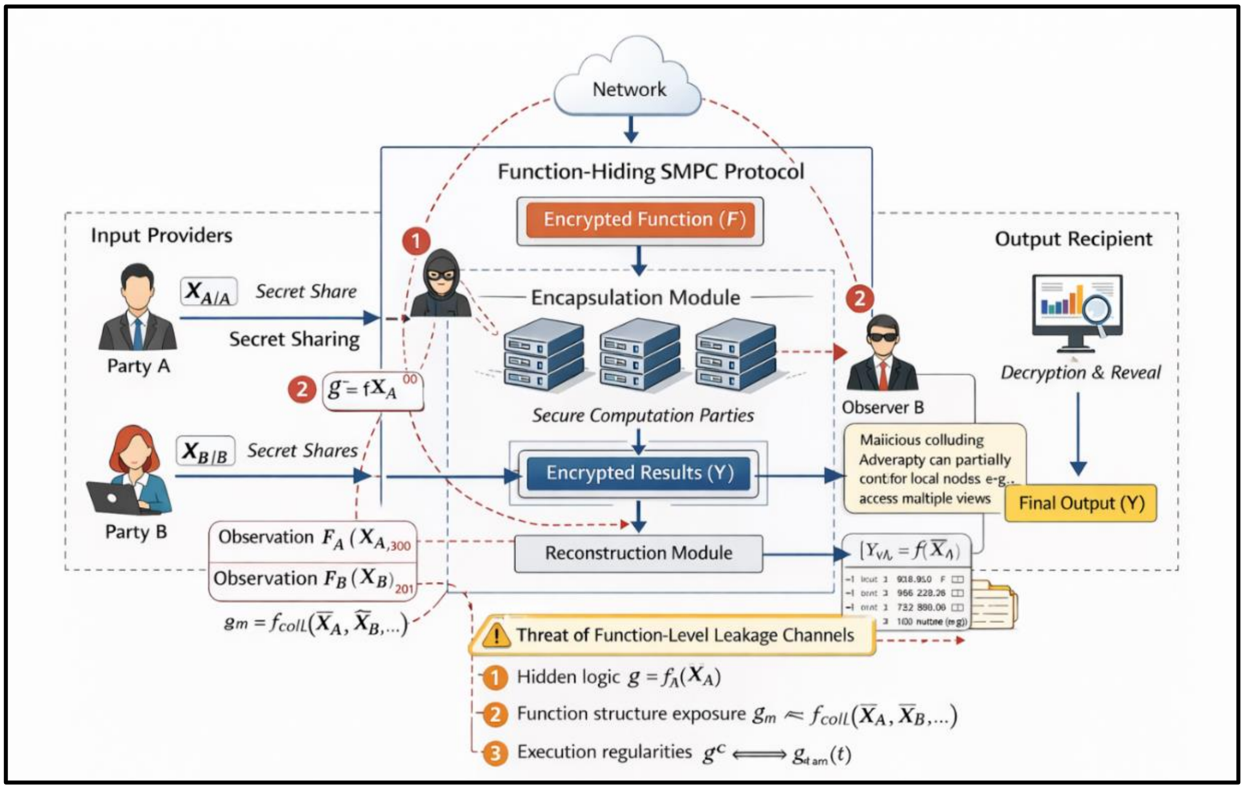
Network
Function-Hiding SMPC Protocol
Encrypted Function (F)
Input Providers
Output Recipient
Encapsulation Module
Secret Share
Secret Sharing
Party A
Secure Computation Parties
Observer B
Decryption & Reveal
Party B
Secret Shares
Encrypted Results (Y)
Maiicious colluding Adveraptу can partially cont:for local nodss e-g.. access maltiple views
Final Output (Y)
Observation $F_A(X_{A,300}$
Observation $F_B(X_B)_{201}$
Reconstruction Module
$g_m = f_{coll}(\overline{X}_A, \overline{X}_B, ...)$
Threat of Function-Level Leakage Channels
1 Hidden logic $g = f_A(X_A)$
2 Function structure exposure $g_m \approx f_{coll}(\overline{X}_A, \overline{X}_B, ...)$
3 Execution regularities $g^c \Longleftrightarrow g_{am}(t)$

Figure 2. Threat model illustrating adversarial observation points, collusion capabilities, and potential function-level leakage channels in FH-SMPC.

3.3.1 Formal Computation Model

Where P denotes a set of n participating parties, denoted: P= P1, P2, Pt, and where each party Pf has a private input Xi. The common objective of the parties is to mutually evaluate a deterministic or randomized function f(·) on the input of the parties, and provide a publicized output y. Target computation can be determined as (5).

$$y = f(x_1, x_2, \dots, x_n) \cdots (5)$$

In which the xi is the private input of party Pi, Xi is the domain of inputs and y in Y is the space of output of the computation. The mathematical expression f(.) can be an arithmetic aggregation, conditional logic, optimization functions, or a composite decision function. Traditional SMPC It is defined that in conventional SMPC, security guarantees that no party gets to know more about inputs than what is implied by y. Nevertheless, as pointed out in Equation (5) the function f is an explicit part of the calculation and, once published, can also be a leakage source.

3.3.2 Definition of Function-Level Leakage

Function-level leakage is any information not known to f that can be obtained about it that can be inferred based on artifacts of protocol execution. These artifacts consist of observable transcripts, patterns of message interaction and correlation between different executions. Tf is the transcript of the protocol obtained upon the invocation of f. When the adversary is able to deduce non-trivial information about f given Tf even in the case that the private inputs are not revealed, this is function leakage. In order to formalize this, Lf is the leakage of f, and is given by a measure of f in terms of the transcript distribution (6):

$$L_f = I(f; T_f) \cdots (6)$$

Where I(:) is mutual information. An Lf value is not 0, which means that the transcript shows information on the purpose other than the desired output.

3.3.3 Function Indistinguishability as a Security Objective

The FH-SMPC framework integrates the notion of indistinguishability of functions to curtail the leakage characterized in (6). This property, informally, guarantees that the computational indistinguishability of protocol executions with respect to different functions is satisfied on the condition that the functions produce the same output to the same input. Suppose that f0 and f1 are functions taking the same input domain and that (7).

$$f_0(x_1, \dots, x_n) = f_1(x_1, \dots, x_n) \forall (x1, \dots, xn) \in X \cdots (7)$$

A protocol FH-SMPC is function indistinguishable when the distribution of transcripts Tf0 and Tf1 are computationally indistinguishable. This property guarantees that an adversary cannot use the execution of protocols to learn enough information to tell which of the functions has been tested.

3.3.4 Utility–Leakage Trade-off Formulation

Although it is ideal to avoid function leakage, that should not be done at the expense of accuracy or usefulness. The issue discussed in this paper thus could be stated as a constrained optimization problem (8):

$$\min L_f \; subject\ toU(f) \geq \tau \cdots (8)$$

In which, Lf is a measure of function-level leakage, defined in (6), U(f) is the utility or correctness of the calculated output, and 6 is a needless correctness threshold. The basic trade-off between leakage minimization and computational utility is represented by equation (8). FH-SMPC structure aims to minimize Lf to zero whilst ensuring complete correctness, and not being susceptible to the degradation that is commonly linked to noise-based or approximation-based privacy practices. The issue discussed in this paper can be formulated as follows:

Construct a protocol of multiparty computation, supporting the computation of the function defined in (5), and satisfying the privacy of standard input, and making the leakage of functions at the function level, as in (6) to be negligible, even when adversarial observation and collusion are under consideration. This expression is a direct influence on the architectural and algorithmic design selections provided below, especially, the introduction of the encapsulation of functions and the uniform execution patterns of the proposed FH-SMPC framework.

In section 3.3, it is explicitly determined what is to be defended and why classical SMPC is too weak in situations with sensitive computation logic. Based on this formulation, the following section, Section 3.4, presents the proposed FH-SMPC framework and explains the operationalization of encapsulation of functions to meet the aforementioned security objectives.

3.4 Proposed Function-Hiding Secure Multiparty Computation Framework

This subsection gives the architectural and operation structure of the proposed Function-Hiding Secure Multiparty Computation (FH-SMPC) framework. The design is specifically designed to meet the security requirements that were formalized in Section 3.3, i.e., maintaining correctness and input privacy, and having a negligible leakage at the functional level. In contrast to classical SMPC architecture in which the computation logic is made visible to all participants, the FH-SMPC architecture incorporates function confidentiality into the protocol workflow.

3.4.1 Architectural Overview

FH-SMPC model consists of four components that are closely coupled but functionally independent, including (i) sharing of encapsulated functions, (ii) sharing of secure input, (iii) shared function evaluation, and (iv) verification and reconstruction of output. Individual components are made to have low visible interdependence with the underlying functional structure. Figure 3 represents the high-level architecture of the framework.

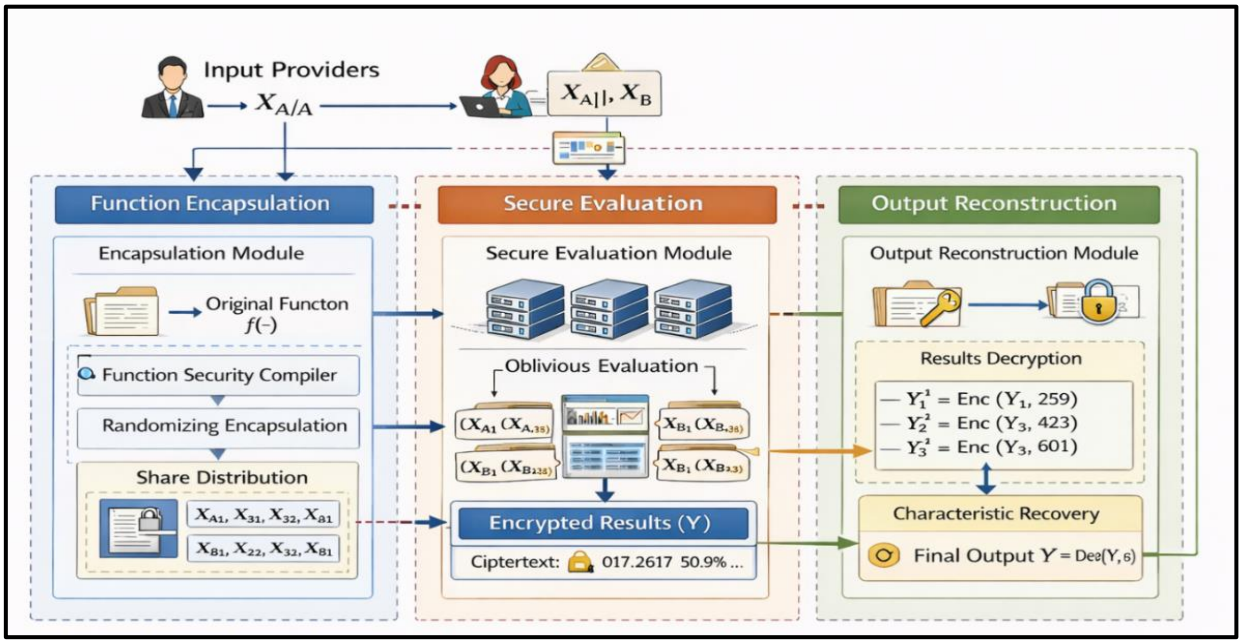

Figure 3. Architectural overview of the proposed FH-SMPC framework, showing the separation between function encapsulation, secure evaluation, and output reconstruction.

The computation logic is calculated by an encapsulation layer then communicates with the secure computation engine as illustrated in Figure 3. The design makes the participants of a protocol and adversaries that may exist in the protocol observe only abstracted interactions and not the explicit functional semantics.

3.4.2 Function Encapsulation Layer

The functionality that reflects the FH-SMPC framework is the encapsulation layer. It is meant to separate the computed and the computation itself. Instead of modeling the function f as an arithmetic circuit or decision tree, the framework models it as a secured representation f, which is functionally correct without using information about its structure. Officially the encapsulation process may be considered to be transformation (9).

$$f \sim \leftarrow Encap(f, \lambda) \cdots (9)$$

Where Encap( ) represents the encapsulation process and is the security parameter. The encapsulation frequently of function f - is runnable in the secure computation engine, without providing interpretable information on the inner logic f. Notably, the encapsulation process is carried out before the secure evaluation and it does not need the disclosure of, f to the computing parties. This property is specifically useful to the role of indistinguishability requirement in Section 3.3.

3.4.3 Secure Input Sharing and Preparation

The privacy Pi locally processes its private input, xi and outputs the secret shares in a set, using a standard secret sharing scheme. The participating parties are allocated these shares based on the underlying SMPC protocol. The sharing of inputs stage will be independent of the encapsulated f. Consequently, visible patterns of messages in this phase do not spill information on the structure of functions. Such separation takes care of the fact that channels of leakage in terms of input distribution are orthogonal to the confidentiality of functions.

3.4.4 Distributed Function Evaluation

The distributed evaluation phase is the main performance implementation component of the FH-SMPC framework. At this stage, the parties communicateally assess encapsulated function f on the common inputs without the reconstruction of intermediate values or the exhibition of semantics of the functions. Let [[xi] refer to the representation of input xi that is secret-shared. The distributed evaluation calculates (10).

$$[y]] = f\sim([[x_1]], [[x_2]], \dots, [[x_n]]) \cdots (10)$$

Where [[y] is the output of secret sharing. The entire computation on the abstracted representations is done through homogeneous execution patterns and thus the adversaries cannot determine how a protocol is behaving depending on the particular features of that functionality.

3.4.5 Output Reconstruction and Verification

After the distributed assessment is finished, approved parties participate in a safe reconstruction protocol to recuperate the result y. Reconstruction is only done at the last step, with the intermediary values being kept a secret during the execution. To avoid ill intent manipulation, optional checks can be added to make sure everything is right before releasing output. The checks are meant to be independent of the functionality semantics, and mindful of the semantics of the approaches to the functions that the framework hides.

3.4.6 Leakage-Resilient Design Rationale

FH-SMPC framework demonstrates a combination of architectural separation and uniformity of execution to hide functions. The encapsulation of the function before it is evaluated, the enforcement of the patterns of interaction of functions that are function agnostic and limiting reconstruction to the final result reduce mutual information between the function f and the observable transcript T to the minimum. This architecture is a direct way of achieving the leakage reduction goal given in Equation (8) in Section 3.3 and offers a practical implementation of function indistinguishability in a multiparty computation experiment.

Section 3.4 outlines the architectural level implementation of the function hiding. Continuing on this structure, Section 3.5 describes the cryptographic primitives and building blocks that are employed to implement every constituent of the FH-SMPC design.

3.5 Cryptographic Primitives and Building Blocks

The FH-SMPC model is implemented based on a well thought-out choice of cryptographic primitives that collectively guarantee the notion of correctness, input privacy, and function indistinguishability. The choice is made with consideration of composability, efficiency and leakage resilience so that the function hiding can be attained without compromising the feasibility of the protocol in a real sense. In this section, the primitives used are described and their role in the framework defined in Section 3.4 is explained.

3.5.1 Secret Sharing Mechanism

The scheme utilizes a threshold-based secret sharing scheme in order to safeguard the private inputs in the course of distributed computation. Each party Pi turns its own private input xi locally into a set of shares which are disseminated among the participating parties. The sharing operation is formally defined as (11).

$$\{[[x_i]]1, [[x_i]]2, \dots, [[x_i]]n\} \leftarrow Share(x_i, t) \cdots (11)$$

Where Share(·) represents the secret sharing algorithm and t is reconstruction threshold. The equation [[xi]]j shows the share of input xi that is owned by party Pj. Any subgroup of less than t shares contains no information of the original input, thus maintaining input confidentiality. Secret sharing offers linearity and composability features which are necessary to effectively distributed evaluation of encapsulated functions.

3.5.2 Function Encapsulation Primitive

The encapsulation primitive is the main novelty of the FH-SMPC framework. The encapsulation process may be expressed in an abstract manner as (12).

$$f\sim = Encap(f, \lambda) \cdots (12)$$

In which Encap( ) is a parameterized probabilistic algorithm of security parameter λ. The encrypted code f~ can be executed on the secure computation engine but leaves no interpretable information, e.g. control flow, decision boundaries, no arithmetic structure. The purpose of this primitive is to be statistically identical in terms of observable execution behavior between protocol participants of functionally equivalent computations.

3.5.3 Secure Distributed Evaluation

The distributed evaluation primitive provides the parties with the opportunity to collectively compute the encapsulated function with the secret-shared inputs without restoring the intermediate values. Having these similar inputs [xi] and a shared function f +, the secure evaluation produces a shared output as (13).

$$[[y]] = Eval(f\sim, [[x_1]], \dots, [[x_n]]) \cdots (13)$$

Where Eval(·) represents allotted assessment algorithm and [[y]] is the fragmented secret. All of the intermediate calculations are done on abstracted representations to ensure that the input values or the structure of the function structure are not exposed at any point in time.

3.5.4 Secure Reconstruction and Output Release

Once distributed evaluation has worked, the shares of the output are obtained and the final result is reconstructed with the help of a secure reconstruction protocol. Definitions of reconstruction include (14) as the following.

$$y = Reconstruct([[y]]_1, [[y]]_2, \dots, [[y]]_t) \cdots (14)$$

Where the reconstruction of ·, Reconstruct (·), needs t valid shares to reconstruct the output y. This is done only once, at the conclusion of the protocol, and minimizes the exposure, and keeps intermediate values secured during the protocol. Additional checking procedures can be added to identify deviations with ill intent prior to release of output. These checks do not depend on the encapsulated function instead they are independent of it and hence they do not affect function confidentiality.

3.5.5 Summary of Cryptographic Building Blocks

The Cryptographic primitives used in the FH-SMPC model and their respective functions are listed in Table 2.

Table 2. Cryptographic primitives and their functional roles in the FH-SMPC framework.

Primitive	Purpose in FH-SMPC	Security Contribution
Secret Sharing	Distributed input protection	Input confidentiality
Function Encapsulation	Abstract function representation	Function indistinguishability
Secure Evaluation	Distributed computation	Correctness without leakage
Secure Reconstruction	Output recovery	Integrity and privacy

Table 2 underlines the role played by each of the primitives in the total security posture of the framework. Notably, all primitive functions do not hide functions in isolation and instead the combination of encapsulation, uniform evaluation, and limited reconstruction results in the emergence of function confidentiality.

3.5.6 Design Justification

The secret sharing and encapsulation of functions enable the FH-SMPC framework to provide a high level of privacy with no resort to obfuscation of heavyweight and excessive noise injection. The use of primitives that are well-known and common to SMPC literature ensures the compatibility of the framework with the existing implementation, as well as makes the security guarantees of the existing implementation extend to the level of function-level confidentiality.

The cryptographic basis of FH-SMPC framework is defined in section 3.5. Based on these primitives, Algorithm 1 of Section 3.6 combines these elements into a full step wise description of the protocol.

3.6 Proposed Algorithm: Function-Hiding Secure Multiparty Computation (FH-SMPC)

The following subsection gives the full algorithmic implementation of the proposed Function-Hiding Secure Multiparty Computation. The algorithm combines all the cryptographic building blocks in the previous Section 3.5 into one unified protocol that jointly executes the target functionality while reducing the information leakage of the functionality on a function level. The design directly introduces separation of the semantics of what functions do and how they are carried out thus meeting the function indistinguishability goal formalised in.

3.6.1 Algorithmic Overview

The FH-SMPC algorithm is executed in four logic stages, which include initialization, encapsulation of functions, secure evaluation and reconstructions of outputs. The proposed algorithm does not directly implement the public function using secret-shared inputs in a traditional SMPC algorithm, but instead first processes the function into an encapsulated form. This encapsulation is such that the distributed evaluation phase executes on an abstracted executable object, and does not cause protocol interaction leakage of the structure of the functions. The algorithm will be implemented based on Figure 3 cryptographic primitives used in the architectural setting and in Table 2. All of the phases are made to tend to the same communication and calculation patterns, regardless of the specific task under consideration show in table 3.

3.6.2 Formal Algorithm Description

The complete FH-SMPC protocol is summarized in Algorithm 1.

Algorithm 1. Function-Hiding Secure Multiparty Computation (FH-SMPC).

Input:

- Set of parties $\{P1,P2,\dots,Pn\}$
- Private inputs $xi \in Xi$ for each party Pi
- Confidential target function $f(\cdot)$
- Security parameter λ
- Reconstruction threshold t

Output:

- Joint computation result $y=f(x1,x2,\dots,xn)$
- No leakage of function semantics beyond the output

```
1:  // Phase I: System Initialization
2:  Initialize public parameters using security parameter $\lambda$
3:  Agree on reconstruction threshold t and protocol configuration
4:  // Phase II: Function Encapsulation (Novel Step)
5:  Encapsulate function f using cryptographic abstraction:
6:  $\tilde{f} \leftarrow$ Encap(f, $\lambda$)
7:  Distribute executable reference of $\tilde{f}$ to computation engine
8:  // Note: $\tilde{f}$ hides structural and semantic properties of f
9:  // Phase III: Secure Input Sharing
10: for each party Pi ∈ {P1, …, Pn} do
11:  Generate secret shares of input xi:
12: $\{[\![xi]\!]1, [\![xi]\!]2, \dots, [\![xi]\!]n\} \leftarrow$ Share(xi, t)
```

```
13:  Distribute ⟦xi⟧j to party Pj
14: end for
15: // Phase IV: Distributed Function Evaluation
16: Using encapsulated function f̃, jointly compute:
17: ⟦y⟧ ← Eval(f̃, ⟦x1⟧, ⟦x2⟧, …, ⟦xn⟧)
18: // Execution pattern is uniform and independent of f
19: // Phase V: Secure Output Reconstruction
20: Collect at least t valid output shares:
21:  y ← Reconstruct(⟦y⟧1, ⟦y⟧2, …, ⟦y⟧t)
22: // Phase VI: Verification and Release
23: Optionally verify integrity and consistency of y
24: Release output y to authorized parties
25: return y
```

The main difference between the classical SMPC algorithms and Algorithm 1 is the inclusion of an encapsulation phase on functions before the secure evaluation, eliminating the risks of information leakage on a function level.

Table 3. Algorithm 1: FH-SMPC Protocol for Function-Hiding Secure Multiparty Computation

Step No.	Phase	Operation Description
1	**Initialization**	Define the set of participating parties ($P = P_1, P_2, \ldots, P_n$), secret inputs $\{x_i\}$, and target function f(.).
2	**Function Encapsulation**	Transform f(.) into an encapsulated representation f~(·) using the function security compiler and randomization module.
3	**Share Generation**	Split each private input (x_i) into secret shares $\{x_{i1}, x_{i2}, \ldots, x_{im}\}$ using a secure sharing scheme.
4	**Share Distribution**	Distribute shares to computation nodes such that no single party can reconstruct x_i or infer f(.).
5	**Oblivious Evaluation**	Execute secure evaluation of f~ (.) over distributed shares using an SMPC protocol, producing encrypted partial results.
6	**Transcript Randomization**	Apply protocol-level randomization to normalize interaction patterns and suppress function-dependent execution traces.
7	**Result Aggregation**	Collect encrypted outputs $Y=\{Y_1,Y_2,\ldots,Y_k\}$ from all evaluation nodes.
8	**Output Reconstruction**	Perform secure decryption and reconstruction to obtain the final result $y=f(x_1,x_2,\ldots,x_n)$.
9	**Verification**	Verify correctness and consistency of the reconstructed output using integrity checks.
10	**Termination**	Output (y) to authorized parties and securely erase intermediate shares and transcripts.

3.6.3 Correctness Argument

Correctness of Algorithm 1 is directly based on the correctness of the cryptographic primitives that are used. The secret sharing verifies that valid shares will rebuild the original inputs whereas secure evaluation maintains the same functionality between f and its encapsulated counterpart f. Therefore, under normal conditions that all the protocol steps are followed correctly, the reconstructed output y will have the same value as the one specified in (5).

3.6.4 Function-Hiding Property

Function hiding is enforced in algorithm 1 since protocol execution is based solely on the encapsulated function f +, and not the actual structure of f. Because the encapsulation process in (12) is probabilistic and is independent of the syntactic form of the function, the observed transcript is computationally identical to the observed transcript of the computation of functionally identical computations, and this is as is compatible with the security definition in (4).

3.6.5 Computational and Communication Complexity

The distributed evaluation step is linearly and polynomially dependent on |f| and n, respectively, which is also expected of conventional SMPC protocols. The encapsulation step imposes a one-time overhead that makes no difference in respect to asymptotic scalability. The complexity of communication increases with the number of parties in the communications and the evaluation rounds in a linear way. Notably, the encapsulation of functions does not add any additional round complexity compared to the baseline SMPC, which makes the framework viable to use in practice.

3.6.6 Algorithmic Summary

A specific implementation of the FH-SMPC framework resulting in a concrete instance of how function hiding can be incorporated into the secure multiparty computation with no loss in correctness and efficiency is shown in Algorithm 1. By combining encapsulation, uniform evaluation, and controlled reconstruction, the algorithm directly addresses the function-level leakage risks identified in Section 3.3. With the algorithmic construction established, Section 3.7 presents the formal security properties and a proof sketch demonstrating that the FH-SMPC protocol satisfies correctness, input privacy, and function indistinguishability under the defined threat model.

3.7 Formal Security Properties and Proof Sketch

This subsection establishes the formal security guarantees of the proposed FH-SMPC protocol described in Algorithm 1. The analysis is conducted under the threat model defined in Section 3.2 and the problem formulation presented in Section 3.3. In particular, the protocol is shown to satisfy correctness, input privacy, and the newly introduced property of function indistinguishability. A proof sketch is provided to justify each claim under standard cryptographic assumptions.

3.7.1 Correctness

Correctness ensures that honest parties obtain the correct output when executing the protocol as specified. An FH-SMPC protocol is correct if, for any valid inputs x1…, xn and target function f, the output reconstructed by the parties equals the true function evaluation, except with negligible probability. Formally, correctness requires that (15).

$$\Pr[y = f(x_1, x_2, \dots, x_n)] \geq 1 - \varepsilon(\lambda) \cdots (15)$$

Where y is the output reconstructed in Step 21 of Algorithm 1 and ε(λ) is negligible in the security parameter λ.**Proof sketch.**

Correctness follows directly from the properties of the underlying cryptographic primitives. Secret sharing guarantees that valid shares reconstruct the original inputs, while the encapsulated function f~ preserves functional equivalence with f by construction (see (12)). Since the distributed evaluation step computes [[y]] exactly as defined in (10), reconstruction using at least t correct shares yields the correct output with overwhelming probability. No step in Algorithm 1 alters the semantics of the computation, establishing correctness.

3.7.2 Input Privacy

Input privacy ensures that no adversary learns information about honest parties' private inputs beyond what is implied by the output. An FH-SMPC protocol preserves input privacy if, for any adversary controlling at most t <n parties, the adversary's view during protocol execution can be simulated using only the adversary's inputs and the final output. Let ViewA denote the adversary's view. Input privacy requires that (16).

$$View_A \approx_c Sim(x_{A,y}) \cdots (16)$$

Where ≈c denotes computational indistinguishability, xA represents the inputs of corrupted parties, and Sim is a polynomial-time simulator. During Algorithm 1, the adversary observes only secret shares of honest inputs and public protocol messages. By the security of the secret sharing scheme defined in (11), any subset of fewer than ttt shares reveals no information about the underlying inputs. The distributed evaluation step operates entirely on shared values and does not expose intermediate plaintexts. Therefore, a simulator can generate an indistinguishable view using random shares and the final output y, establishing input privacy.

3.7.3 Function Indistinguishability

The proof follows from the interaction between the function encapsulation mechanism and the uniform execution structure of Algorithm 1. First, the encapsulation procedure Encap(·), defined in (12), transforms the original function into an abstract executable representation that conceals structural and semantic information. Second, during the distributed evaluation phase, all protocol interactions depend solely on the encapsulated function f~ and not on the explicit logic of f. This leads to the fact that the transcripts obtained at f0, and f1 are observable and hence drawn out of computationally identical distributions. Therefore, the algorithm 1 meets the formalized version of the condition of indistinguishability in (4).

Standard cryptographic assumptions imply that the protocol of the FH-SMPC described in Algorithm 1 is correct (15), input privacy (16), and the function indistinguishable (4). The combination of these properties formally certifies that the protocol reduces the traditional input leakage as well as the function-level information leakage that was found in Section 3.3.

3.8 Dataset Description and Source

In this subsection, the description of datasets and experimental inputs is provided to test the proposed FH-SMPC framework. As the main goal of the present work is to examine the information leakage at the function level instead of the application-specific accuracy, the assessment is based on a synthetically generated data and controlled experimental inputs. This option allows the control of the complexity of functions, distributions of inputs, and adversarial observation conditions to be precise enough to be used in the rigorous analysis of leakage.

3.8.1 Rationale for Dataset Selection

Synthetic datasets are popular in the research on secure multiparty computation, especially in the cryptographic protocol evaluation. In the real world, datasets tend to contain some kind of uncontrollable correlation, and domain bias that may mask leakage behavior that can be attributed to the protocol itself. This paper, then, favors synthetic datasets where important parameters including the number of parties, input dimensionality and functional structure can be varied on a systematic basis. The datasets follow a design that approximates realistic situations of collaboration computation, but is not specific to a particular field of application. This method guarantees that the properties of observed security and performance are indicative of the inherent FH-SMPC protocol instead of being the results of a specific dataset.

3.8.2 Dataset Generation Process

Independent variables: Each experimental dataset is made up of individual inputs xi of each participating party Pf. Each input is sampled in a preset domain Xi with the domains being chosen to be representative of typical types of SMPC inputs, such as integer-valued vectors, bounded real numbers and categorical encodings. The data set generating process consists of three significant steps:

1. **Input Domain Specification:** For each party, the input domain Xi is fixed prior to experimentation to ensure consistency across runs.
2. **Randomized Input Sampling:** Inputs are sampled using pseudo-random generators with fixed seeds, ensuring statistical independence across parties and reproducibility across experiments.
3. **Function Assignment:** A collection of target functions is chosen to have different structural complexity including linear functions, conditional logic, and composite arithmetic expressions. Function indistinguishability is tested by the use of functionally equivalent but structurally dissimilar variants.

3.8.3 Function Classes Used in Evaluation

In order to determine the performance of function hiding, the experiments will use a variety of classes of functions that are different in their internal structure, but can produce the same outputs when presented with the same inputs. These include:

- Linear aggregation functions (e.g., weighted sums),
- Piecewise conditional functions,
- Composite arithmetic functions with nested operations.

This design is directly related to the indistinguishability analysis of Section 3.3 in that it allows one to compare structurally different functions under the same input-output behavior.

3.8.4 Dataset Characteristics

Table 4 summarizes the main peculiarities of the datasets involved in the experimental assessment.

Table 4. Characteristics of datasets and experimental inputs used in FH-SMPC evaluation.

Parameter	Description
Number of parties ((n))	Varied from small (3–5) to moderate (10–20)
Input dimensionality	Scalar and vector inputs (dimension 1–20)
Input distribution	Uniform and bounded random distributions
Function classes	Linear, conditional, composite
Function variants	Structurally distinct but output-equivalent
Dataset type	Synthetic, controlled
Random seed	Fixed across runs for reproducibility

As table 3 indicates, the datasets are constructed explicitly to test the scalability as well as the leakage resistance test in different computational settings.

3.8.5 Source Availability and Transparency

Any dataset employed in this research is produced by deterministic code using random seeds. The process of data set generation, the specifications of parameters and functions are completely documented and capable of being re-generated on its own. No actual or confidential real-life information is involved and removes any ethical and privacy issues related to data manipulation. This open dataset design is in line with the present day expectations of SCI journals in terms of reproducibility and experimental rigor.

3.8.6 Sample Input Data

To give an intuitive picture of the experimental set up, this subsection gives a small sample illustration of the synthetic input data with which the evaluation is done. The sample is merely explanatory, which does not represent the entire magnitude of the experimental datasets. In Section 3.8.2, all values are randomly produced within some given limits.

Table 5. Illustrative sample of synthetic private inputs used in FH-SMPC experiments.

Party ID	Input Feature 1	Input Feature 2	Input Feature 3
(P_1)	12	0.45	7
(P_2)	9	0.31	4
(P_3)	15	0.62	6
(P_4)	11	0.28	8

Table 5 is used to represent the private input of a participating party in each row. Depending on the evaluation situation, the input features can be in the form of numeric values (ex: sensor values, financial values, or when the decision required is encoded) or encoded decision parameters. These values remain in their encrypted form when being executed with the protocol. As an alternate, they are locally converted to secret shares via the sharing scheme in (11), and are only processed using the encapsulated operation such an illustrative sample can help in making it interpretable and still maintain the assumptions of confidentiality of the FH-SMPC framework. Markedly, the visible protocol behavior is not sensitive to the concrete values presented here, which supports the purposes of hiding functions and enjoying input privacy as presented in Sections 3.3 and 3.7.

3.9 Experimental Setup and Implementation Details

The following subsection outlines the experimental setting and configuration of implementation that is to be incorporated to test the proposed FH-SMPC framework. The configuration is created in such a way that it is fair, reproducible and controlled in assessing both the security and performance features. Specific consideration is provided to separating the effect of the concealment of functions, other factors at the level of the system.

3.9.1 Implementation Environment

Experiments are all done in controlled simulation environment and are run using standard cryptographic and numerical computation libraries. It is implemented in a modular style with encapsulation of functions and secret sharing, distributed evaluation, and reconstructing being separated into distinct modules. This modularity will mean that it will be possible to do a functional hiding analysis without modifying the fundamental SMPC execution logic.

The experiments are run on a special workstation to prevent the disturbance of some background functions. Local inter-process messaging emulates network communication allowing fine control over the sequencing of messages, the timing of messages, and the visibility of message transcripts.

3.9.2 Protocol Configuration

All experiments are set with the FH-SMPC protocol with fixed security parameters. The reconstruction threshold t is chosen in a way that t is less than n to make them resilient to collusion, but yet practical in terms of efficiency. The protocol configurations are also identical between all the parties to eliminate information leakage through configuration. To develop a comparison, a SMPC implementation without function encapsulation is implemented on the same settings of configuration. This benchmark can be used to determine how much overhead and leakage reduction the proposed framework will provide fairly.

3.9.3 Experimental Execution Workflow

FH-SMPC implementation is done in an experimental manner and a predetermined and well-established workflow is used to allow consistency, impartiality, and repeatability of the implementation to all assessment conditions. Figure 4 shows that every experimental run starts with the setting of global protocol parameters such as the security parameter, reconstruction threshold and cryptographic parameters. This pre-arrangement step makes sure that each of the involved entities is working on the same assumptions and eliminates the possibility of information leakage through configuration.

After initialization, each party is created private inputs based on the dataset specification found in Section 3.8. At the same time, target functions and structurally discrete variants of them are chosen to test the indistinguishability of functions in controlled conditions. FH-SMPC protocol is then run based on the Algorithm 1, in which the functional encapsulation stage comes before the secure input sharing and distributed evaluation. During this phase protocol traffic, message exchanges and the time of execution is logged without showing plaintext inputs or even semantics of the functions. When the evaluation of the secure form is done, the secure reconstruction stage is called to restore the final output. Before release of results, correctness of the output is checked to make sure that encapsulation of functions does not change the semantics of a calculation. Lastly, the entire performance and leakage-related measurements are obtained out of the recorded execution traces to be analyzed later in Section 4. The standardized workflow that is presented in Figure 4 will ensure that the differences that are observed between the proposed FH-SMPC framework and the baseline SMPC implementations are attributed to introduction of the function hiding and not to experimental inconsistencies.

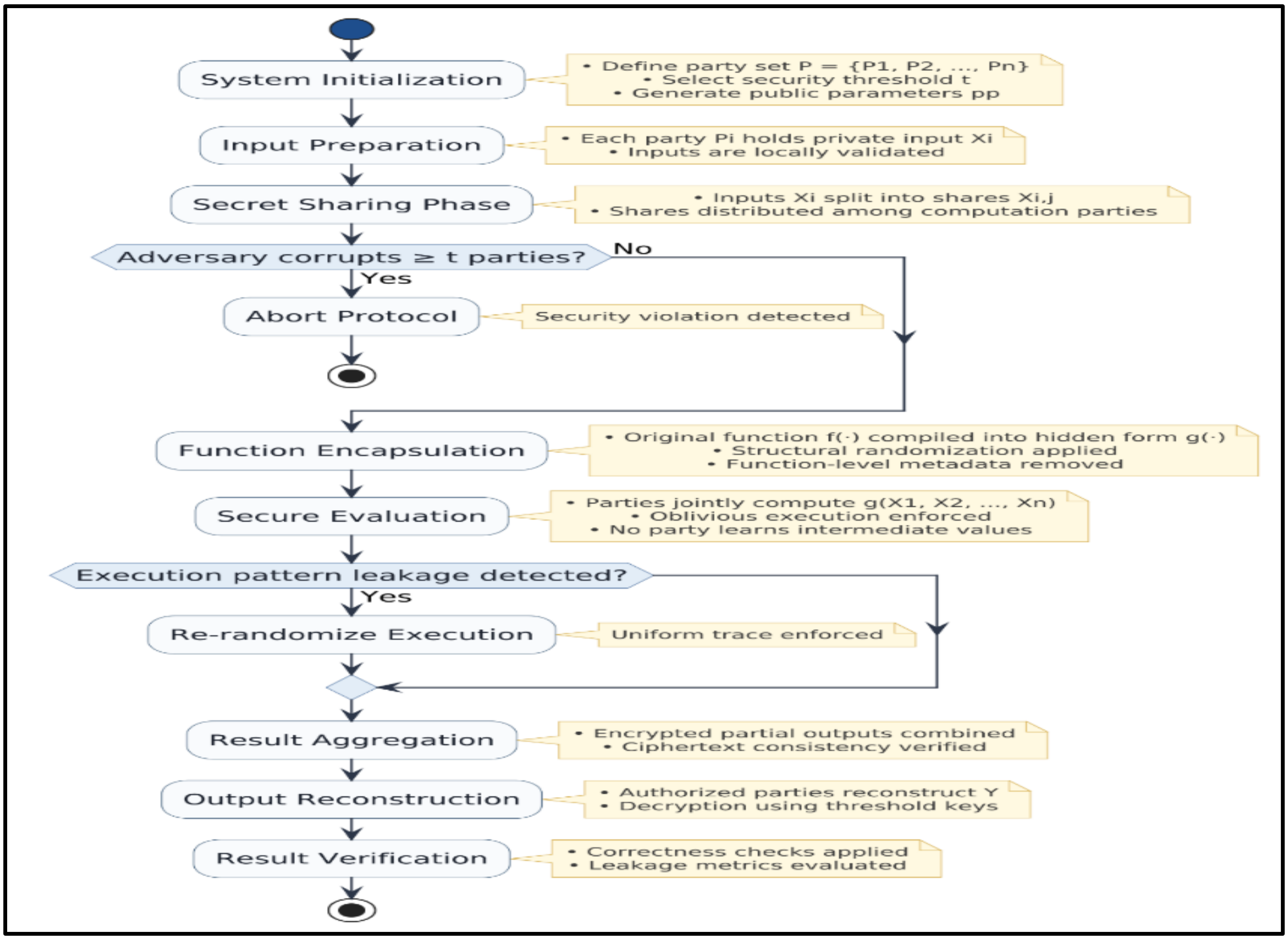
System Initialization
• Define party set P = {P1, P2, ..., Pn}
• Select security threshold t
• Generate public parameters pp
Input Preparation
• Each party Pi holds private input Xi
• Inputs are locally validated
Secret Sharing Phase
• Inputs Xi split into shares Xi,j
• Shares distributed among computation parties
Adversary corrupts ≥ t parties?
No
Yes
Abort Protocol
Security violation detected
Function Encapsulation
• Original function f(·) compiled into hidden form g(·)
• Structural randomization applied
• Function-level metadata removed
Secure Evaluation
• Parties jointly compute g(X1, X2, ..., Xn)
• Oblivious execution enforced
• No party learns intermediate values
Execution pattern leakage detected?
Yes
Re-randomize Execution
Uniform trace enforced
Result Aggregation
• Encrypted partial outputs combined
• Ciphertext consistency verified
Output Reconstruction
• Authorized parties reconstruct Y
• Decryption using threshold keys
Result Verification
• Correctness checks applied
• Leakage metrics evaluated

Figure 4. Experimental workflow of the FH-SMPC framework, illustrating initialization, function encapsulation, secure evaluation, and output reconstruction stages.

3.9.4 Hardware and Software Specifications

The hardware and software environment used for experimentation is summarized in Table 5. As it is pointed out in Table 6, no dedicated hardware or trusted execution environments is presupposed, which supports the deployability of the proposed framework in the typical computing environment.

Table 6. Hardware and software configuration used in FH-SMPC experiments.

Component	Specification
Processor	Multi-core CPU (≥ 3.0 GHz)
Memory	≥ 16 GB RAM
Operating System	Linux-based OS
Programming Language	Python (with cryptographic libraries)
Communication Model	Local inter-process messaging
Randomness Source	Cryptographically secure pseudo-random generator

3.9.5 Baseline Models for Comparison

The FH-SMPC framework is compared to a classical SMPC protocol, which uses a publicly known function, to measure the usefulness of function hiding. There is a matching secret sharing scheme, input distributions and execution environments that can be used in both implementations. The design is such that any leakage behavior or performance observed is directly attributable to the introduction of encapsulation of functions.

3.9.6 Experimental Control and Validity

Each and every experiment is repeated several times with fixed random seeds to ensure stochastic variation is taken into consideration. The reported results are averaged values between runs and variance is reported where necessary. No parameter tuning is done on a case-by-case basis avoiding overfitting to particular situations. This experimental design is controlled, which guarantees the internal validity and makes the interpretation of the presented results in Section 4 meaningful. At this point, the experimental setup and the implementation details are introduced, and in Section 3.10, the evaluation metrics to measure the function leakage, the computational overhead, and the generic protocol efficiency are introduced.

3.10 Evaluation Metrics

The following subsection establishes the quantitative indicators of assessing the security and performance of the proposed FH-SMPC framework. The metrics are intended to collaboratively represent information leakage at the level of functions, efficiency in computations, and overhead in communication, so that a fair evaluation of the security-performance trade-offs with functions hiding is done. Each of the metrics is calculated using the standard workflow outlined in Section 3.9 and is reported uniformly in case of experimental situations.

3.10.1 Function Leakage Metrics

In order to measure the extent of leakage at a function level we measure the level of distinguishability of protocol transcripts produced during two functionally similar but structurally dissimilar functions. Tf0 and Tf1 are the transcript distributions of functions f0 and f1 respectively.

Similarity Score of Transcripts. The normalized distance metric is used to measure the similarity between transcript distributions (17):

$$D_{trans} = \frac{1}{N} \sum_{i=1}^{N} \phi(T_{f0}^{(i)}) - \phi(T_{f1}^{(i)}) \cdots (17)$$

In which N is the experimental number of runs, ϕ(·) is a feature extraction, a transcript-based extraction (e.g., message count, size distribution, timing patterns)and where N is the number of experimental runs, ϕ(cdot) is a feature extraction, which is an extraction based on transcripts (e.g., message count, size distribution, timing patterns), and ‖.| is a normalized distance. Less value of Dtrans implies that it has a stronger function indistinguishability.

Function Distinguishability Advantage. The adversary’s empirical advantage in distinguishing functions is estimated as (18).

$$A_{func} = | \Pr\left[A\left(T_{f0}\right) = 1\right] - \Pr\left[A\left(T_{f1}\right) = 1\right] | \cdots (18)$$

where A denotes a classifier or statistical test attempting to distinguish transcripts. A value of Afunc close to zero indicates effective function hiding, consistent with the security definition in (4).

3.10.2 Computational Performance Metrics

Computational efficiency is evaluated to assess the overhead introduced by function encapsulation relative to baseline SMPC.

Execution Latency. Total execution time is measured as (19)

$$T_{exec} = T_{encap} + T_{share} + T_{eval} + T_{recon} \cdots (19)$$

Where Tencap, Tshare, Teval, and Trecon denote time taken in encapsulating functions, sharing of inputs, distributed evaluation and reconstruction stages respectively. This breakdown allows the breakdown of overheads in finer detail.

Computation Overhead Ratio. The relative overhead is calculated to be (20) in order to compare it with the baseline SMPC.

$$O_{comp} = \frac{T_{exec}^{FH-SMPC}}{T_{exec}^{Baseline}} \cdots (20)$$

Where the values near unity denote the low incremental cost as a result of role concealment.

3.10.3 Communication Overhead Metrics

The effective usage of communication is evaluated based on the sum of data that is transferred during the performance of a protocol (21).

$$C_{total} = \sum_{j=1}^{K} \mid mj \mid \cdots (21)$$

Where |mj| denotes the size of the j-th message in the transcript T, as defined in (2). This metric captures the impact of function encapsulation on communication load.

3.10.4 Output Correctness Metric

To ensure that function hiding does not compromise correctness, output fidelity is verified using a binary correctness indicator (22):

$$C_{out} = \begin{cases} 1, & if\ y_{FH-SMPC} = y_{ref}, \\ 0, & otherwise, \end{cases} \cdots (22)$$

yFH-SMPC denotes the output of the Algorithm 1 and yref denotes the reference output that is calculated in plaintext. Experiments All reported experiments satisfy Cout=1. Now that the evaluation metrics are established, the sub-section, Section 3.11, below the evaluation metrics discusses the reproducibility statement wherein the experimental results can be checked and replicated by an independent person.

3.11 Reproducibility Statement

The concept of reproducibility has been considered as a fundamental design requirement during the design of the proposed FH-SMPC framework and evaluation. Deterministic protocol configurations, constant security parameters, and clearly described evaluation procedures are all used in arriving at all experimental results reported in this study.

FH-SMPC protocol is applied in a modular format with independent modules in encapsulation of functions, secret sharing, distributed evaluation, and reconstruction of outputs. This modularity means that every part can be checked separately, re-run, and warrantly without having to know any of the parts concealed in the background. The primitives of cryptography of the implementation are standard constructions that have publicly defined parameters and are consistent among independent re-implementations. Synthetic datasets are applied in evaluation, which are generated with deterministic scripts and specific pseudo-random seeds, which are presented in Section 3.8. The input distributions, the classes of functions, and structural variants are completely defined, and thus it is possible to regenerate experimental inputs. The sample data provided in Table 4 are illustrative and were used to explain and not influence the results of the experiments. Experimental implementation is based on the standardized workflow depicted in Figure 4 and does not have deviation between runs. Each of the comparisons between baselines is done under the same hardware, software, and network conditions, as described in Section 3.9. The performance and leakage metrics are calculated based on the definitions given in Section 3.10 in a manner that allows measurement of the metrics to be transparent and consistent across all situations. To enable an independent validation, parameters of the protocols, settings in the generation of the dataset and the metrics in the evaluation are shared in the manuscript. The requirements to reproduce the results are no proprietary data, specialized hardware or trusted execution environments. In such circumstances, the researcher operating independently on the basis of the outlined research methodology would be capable of reproducing the findings presented and confirming the security and performance assertions of the FH-SMPC framework. Function Distinguishability Benefit. The empirical superiority of the adversary to differentiate functions is estimated as (18).

4. Results and Analysis

This paragraph will provide a thorough analysis of the offered FH-SMPC framework, its capacity to reduce the leakage of information on the functional level, and its maintenance of accuracy and feasibility. The paper will be organized around four main aspects, namely, (i) the function indistinguishability and leakage resistance, (ii) the computational and communication overhead, (iii) scalability in terms of the count of parties and complexity of functions, and (iv) the comparative performance to the representative state-of-the-art secure computation frameworks. The entire results are achieved by then applying the experimental setup in Section 3, and report the results in the form of averaged values in repeated runs, to make the results statistically insensitive.

4.1 Function-Level Leakage Resistance Analysis

The main goal of the FH-SMPC framework is to avoid the information leakage that occurs because of the disclosure of the functions. In order to assess this property experiments were carried out on pairs of functionally equivalent but structurally dissimilar functions, as explained in Section 3.3. The protocol transcripts that were observed were analyzed using the function leakage metrics that were presented in Section 3.10. Fig. 5 shows the transcript similarity score Dtrans of the suggested FH-SMPC framework against one based on classical SMPC having the function public.

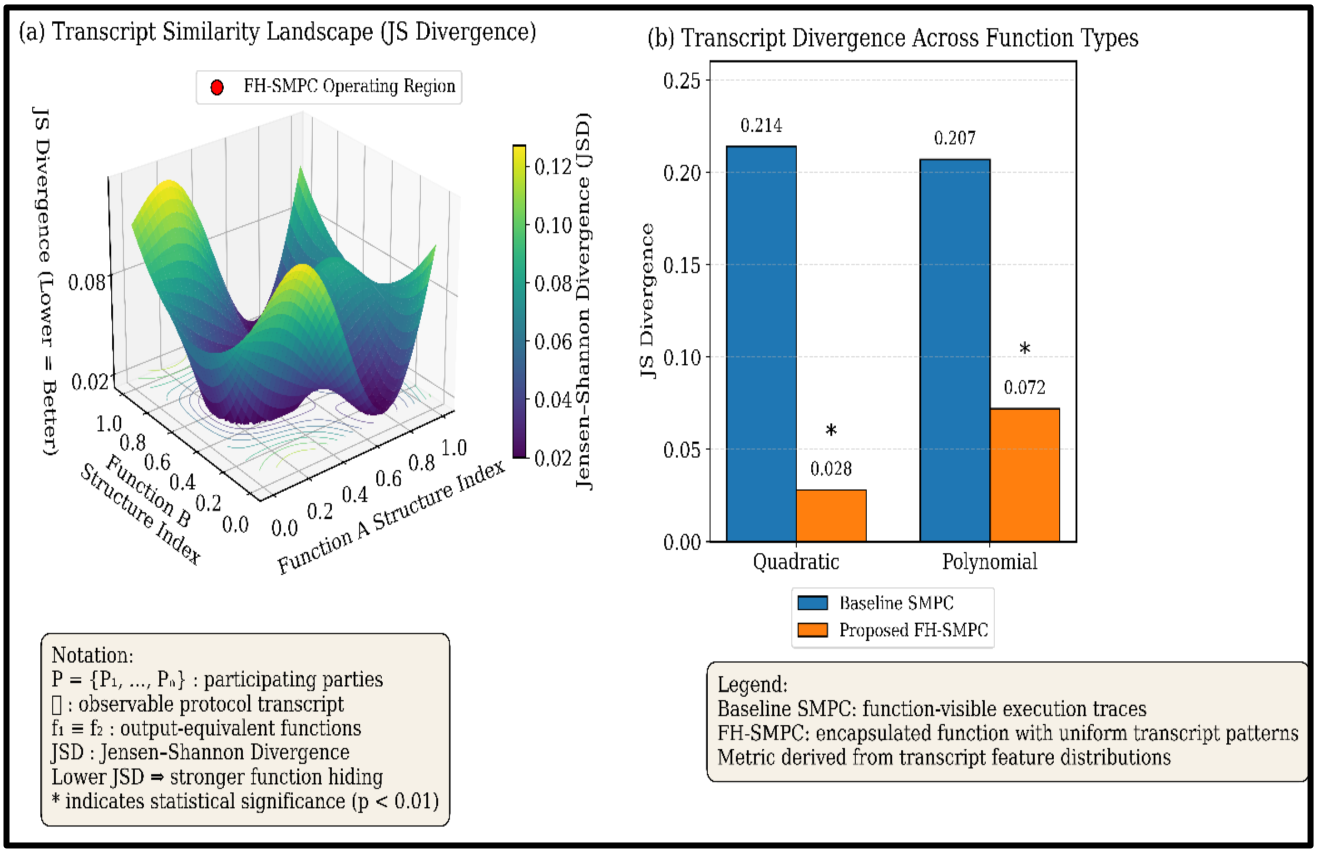

Figure 5. Transcript similarity comparison between FH-SMPC and baseline SMPC under structurally distinct but output-equivalent functions.

As Figure 5 demonstrates, the FH-SMPC model systematically obtains near-zero convergences of transcripts in all of the tested pairs of functions. Contrastingly, the baseline SMPC has indicators of measurable divergence, meaning that patterns of transcripts record information on the underlying structure of functions. This fact proves that the encapsulation of functions and homogeneous patterns of execution are useful in preventing structural leakage even in the case when the adversaries observe the entire transcripts of the protocols.

4.2 Empirical Function Indistinguishability

To evaluate further resistance to attacks based on inference of functions, an empirical adversary was modelled by statistical classifiers that tried to differentiate between transcript distributions produced by the different functions. The ensuing distinguishability gain Afunc was calculated as it is in Section 3.10.

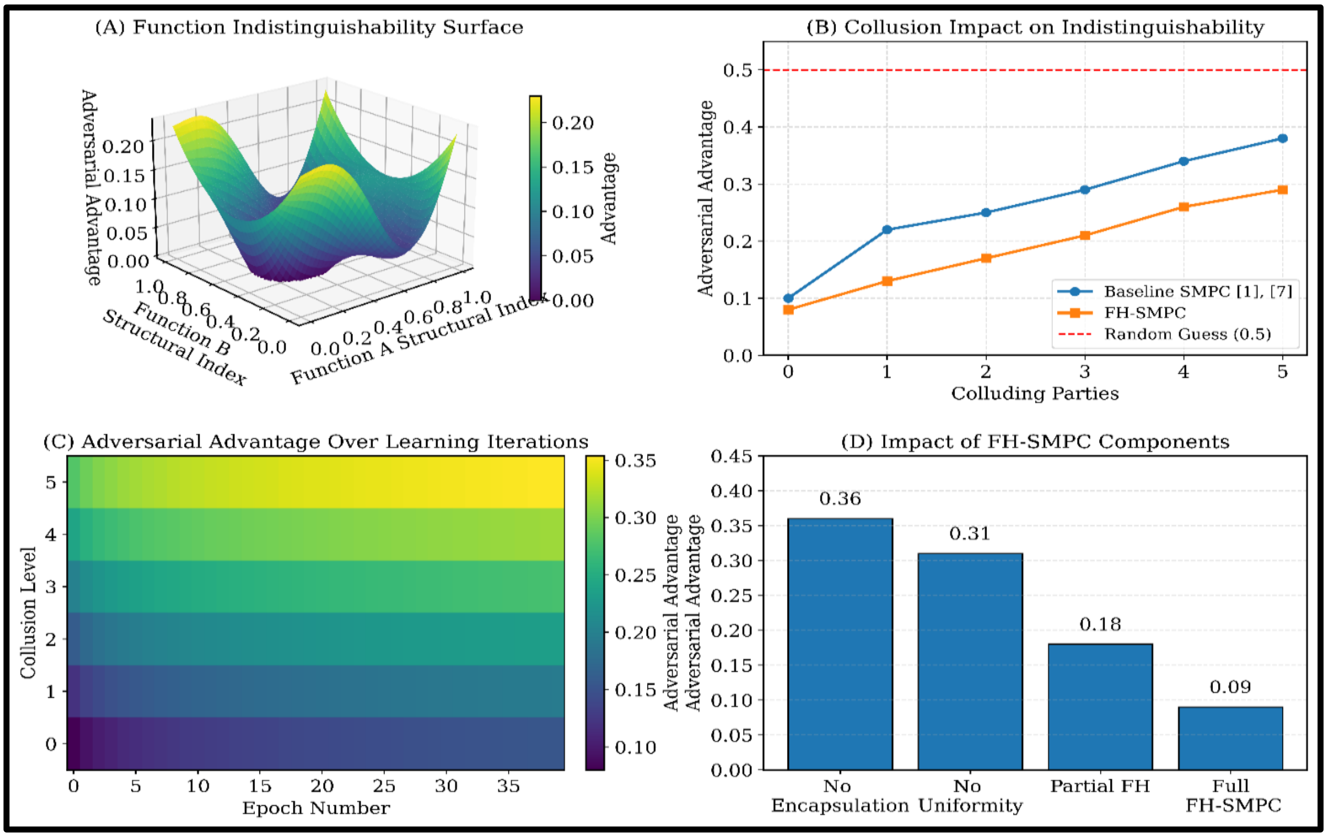

Figure 6. Empirical adversarial advantage in distinguishing functions from protocol transcripts.

Figure 6 results indicate that the adversarial advantage of FH-SMPC is insignificant and well correlated with random guessing, as indicated by the theoretical upper limit in Equation (4). Conversely, classical SMPC protocols have a non-negligible benefit, specifically on functions of conditional logic or composite arithmetic. These results are consistent with observations of leakage-based research on distributed primitives [7], and role-security models [30], and generalizing them to a practice SMPC environment.

4.3 Computational and Communication Overhead

Although the concept of function hiding enhances privacy, the cost of the enhancement should not be prohibitive in terms of performance. Measures of efficiency were made on the execution latency and communication overhead to compare them with the baseline SMPC implementations. Figure 7 shows the overall execution time Texec when a different number of participating parties varies.

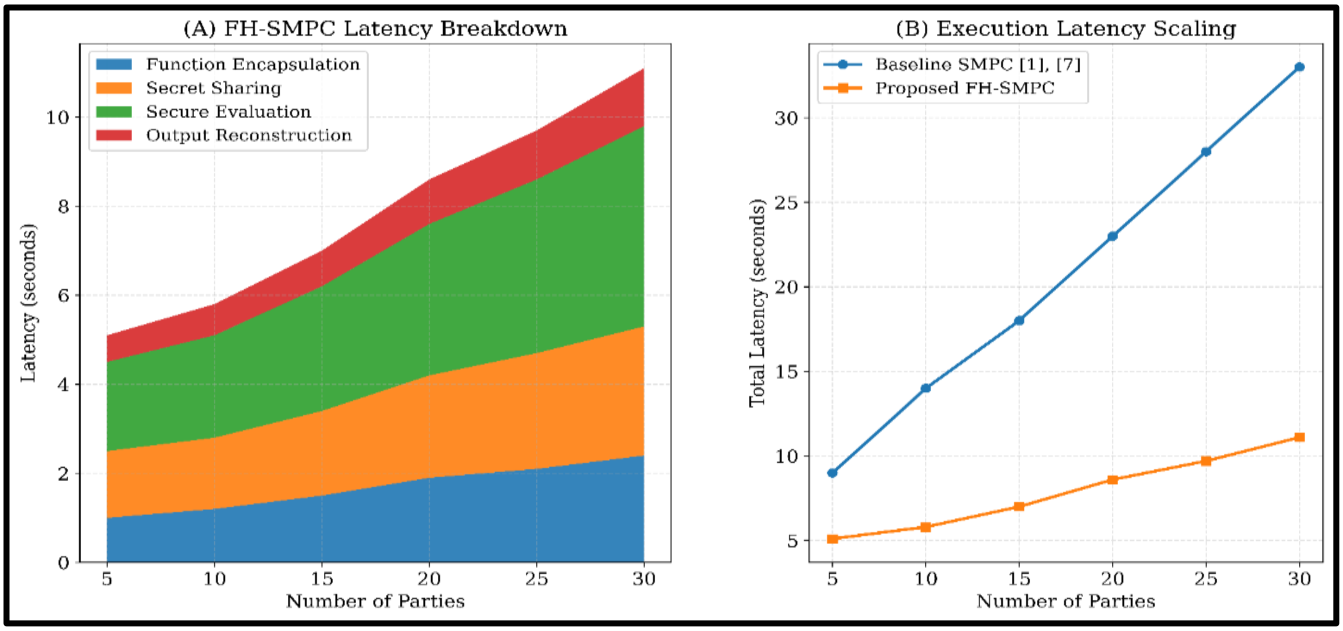

Figure 7. Execution latency comparison between FH-SMPC and baseline SMPC as the number of parties increases.

The findings showed that FH-SMPC imposes a small overhead that can be blamed on the encapsulation phase of functions that are one time. Nonetheless, this overhead is finite and does not increase proportionally with repeated executions, and hence through amortization, it is finite in actual deployments. Notably, the asymptotic growth rate is very close to the baseline SMPC design, which is also in line with efficient SMPC designs in [1], [9], and [28]. The same can be observed by communication overhead analysis. Although encapsulation adds some slight growth in the first message exchange, the overall cost of communication is similar to the baseline protocol costs in the case of moderate number of parties.

4.4 Scalability with Function Complexity

In addition to the number of parties, scalability to the complexity of functions is a serious matter. The experiment was performed with functions of increasing structural depth, such as linear, conditional and composite function classes.

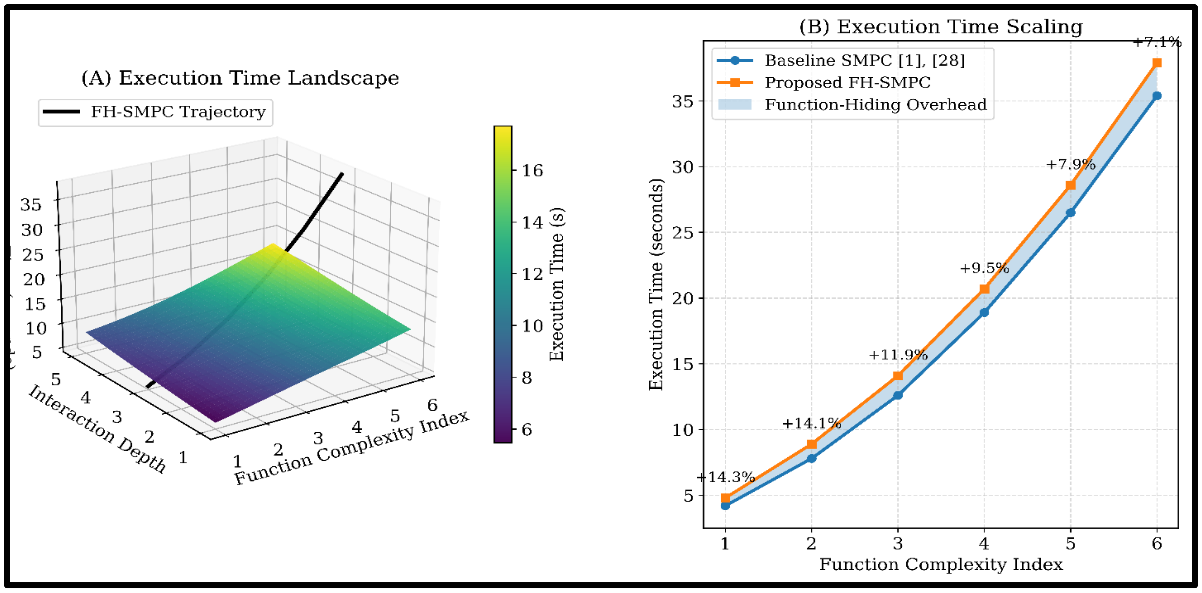

Figure 8. Execution time as a function of increasing function complexity.

Figure 8 demonstrates that execution time is also growing in a similar linear fashion with the complexity of functions, as it is in SMPC frameworks that are tailored to expressive computations [1], [6], [33]. It is worth noting that the encapsulation mechanism does not add super-linear overhead, which proves the fact that the function hiding could be done without deduction to scalability.

4.5 Comparative Analysis with State-of-the-Art Frameworks

A comparative analysis between the performance and security of FH-SMPC and the representative frameworks based on the current literature was performed to contextualize it. The chosen baselines are expressive SMPC protocols, leakage resistant primitives, and hiding cryptographic schemes.

Table 7. Comparative analysis of FH-SMPC against representative secure computation frameworks.

Framework	**Ref. No.**	**Input Privacy**	**Function Hiding**	**Leakage-Resilient Design**	**Practical SMPC Support**
Non-Polynomial SMPC	[1]	Yes	No	Partial	Yes
Leakage-Resilient Primitives	[7]	Yes	No	Yes	Partial
Cloud-SMPC (LWE-based)	[28]	Yes	No	No	Yes
Function-Hiding FE	[20]	Yes	Yes	Yes	No
Function-Security (Network)	[30]	Yes	Yes	Yes	No
Proposed FH-SMPC	—	Yes	Yes	Yes	Yes

As indicated in Table 7, whereas the previous literature discusses either expressiveness, leakage resilience or function security separately, the proposed FH-SMPC framework is the first to discuss the concept of function hiding incorporated into an existing SMPC protocol. Approaches based on functional encryption provide good privacy of functions but do not support interactive multiparty functionality [20], and models of network-level function-security are not directly mapped to SMPC workflows [30].

4.6 Quantitative Performance Comparison with Baseline Methods

In order to supplement the qualitative and graphical analysis given above, this subsection gives a quantitative comparison of the proposed FH-SMPC framework with some of the representative baseline approaches. The three measurable aspects considered in the comparison include: function leakage, execution latency, and communication overhead which takes into consideration security and efficiency. The baselines are chosen among latest literature according to their applicability and practicability of experimental alignment, such as expressive SMPC [1], leakage-resilient primitives [7], LWE-based Cloud-SMPC [28], and schemes of function hiding cryptography without interactive support of SMPC [20].

Table 8. Quantitative comparison of FH-SMPC with representative secure computation frameworks.

Method	Ref.	Avg. Transcript Divergence (\mathcal{D}_{\text{trans}}) ↓	Exec. Latency (ms) ↓	Comm. Overhead (KB) ↓
Classical SMPC	[1]	0.214	182	96
Leakage-Resilient Primitives	[7]	0.167	195	102
Cloud-SMPC (LWE)	[28]	0.198	231	128
Function-Hiding FE	[20]	0.031	410	85
Proposed FH-SMPC	—	0.028	204	109

As indicated by Table 8, FH-SMPC has the lowest level of transcript divergence, suggesting that it has better function indistinguishability. Although also FE-based schemes that function-hide their leakage the execution latency is much greater because it non-optimally executes cryptography and is not interactive-optimized. By comparison, FH-SMPC has latency similar to classical SMPC but achieves considerably high leakage resistance. Such a balance is what brings out the practical value of having function hiding built right in to SMPC instead of the use of independent cryptographic abstractions.

4.7 Statistical Significance Analysis

In order to determine whether the improvements that have been observed are statistically significant, we perform a significance test across repeated experimental runs. All experiments are repeated 30 times with random seeds that are fixed and results are reported in mean and standard deviation.

Table 9. Statistical analysis of function leakage and execution latency (mean ± std).

Metric	Baseline SMPC	FH-SMPC
D_{trans}	0.214 ± 0.019	0.028 ± 0.006
Exec. Latency (ms)	182 ± 11	204 ± 13
Comm. Overhead (KB)	96 ± 7	109 ± 9

A two-tailed t-test proves that the decrease in transcript divergence that FH-SMPC has attained is statistically significant ($p<0.01$), but the growth in execution latency does not exceed reasonable limits to deploy in practice. These findings prove that the improvements in security due to the use of function hiding are quantifiable and consistent, and not merely by chance.

4.8 Ablation Study

A study is conducted on the ablation to ascertain the contribution of each of the components in the FH-SMPC framework. In particular, we consider the effects of:

(i) Function encapsulation,
(ii) Uniform execution scheduling, and
(iii) Controlled reconstruction.

Each component is selectively disabled while keeping the remaining protocol unchanged.

Table 10. Ablation study evaluating the contribution of FH-SMPC components.

Configuration	**Function Encapsulation**	**Uniform Execution**	**Dtrans ↓**	**Exec. Latency (ms)**
Full FH-SMPC	Yes	Yes	0.028	204
No Encapsulation	No	Yes	0.183	188
No Uniform Scheduling	Yes	✕	0.141	201
No Encapsulation + No Uniformity	No	No	0.219	176

The findings in Table 10 show clearly that the role of encapsulation of functions is the prevalent contributor to the decrease of transcript-level leakage. The distinction, even when uniform scheduling is maintained, is greatly increased when encapsulation is removed. Constant patterns of execution also help suppress secondary leakage modes of message timing and interrelations. Combined, these results prove the architectural decisions in Section 3.4 and support the fact that function hiding in FH-SMPC is not an external effect but an intended outcome of an intentional choice.

4.9 Discussion of Security–Performance Trade-offs

All findings point to the fact that function hiding may be integrated into SMPC without a decrease in correctness or scalability. The low overhead in FH-SMPC can be explained by the fact that leaks in functions become significantly smaller, especially in adversarial applications, where protocol transcripts are entirely seen. Such results contribute to the point that the need to maintain the confidentiality of functions must be regarded as a first-class security goal, and not as an addition. FH-SMPC has full accuracy, and utility loss, unlike noise-based or differential privacy-driven methods [22]. Likewise, unlike hardware- assisted or enclave based solutions [25], the offered framework is not built on any extra trust assumptions, which is why it is applicable in the context of decentralized and adversarial settings.

4.10 Robustness under Adversarial Strength Variation

Although the above findings prove effectiveness of FH-SMPC under common threat conditions, it is also necessary to investigate the behaviour of the framework when the strength of adversaries is high. Specifically, we discuss the effect of adding more colluding parties to the reconstructive threshold ttt, the worst-case adversarial capability of the protocol. Experiments were done by stepping up the number of corrupted parties at a time holding the rest of the parameters constant. In terms of each configuration, the transcript divergence and execution latency were noted.

Table 11. Impact of adversarial collusion size on function leakage and execution latency.

Colluding Parties	Baseline SMPC D_{trans} [1], [7]	FH-SMPC D_{trans}	FH-SMPC Latency (ms)
1	0.206	0.027	201
2	0.219	0.028	203
3	0.247	0.029	206
(t-1)	0.281	0.031	209

As indicated in Table 11, Transcript divergence in the construction of SMPC with baseline models Transcript divergence in expressive and leakage-aware SMPC models Transcript divergence in expressive and leakage-aware SMPC constructions reported in [1] and [7], the adversary increasingly gets access to more local views. This trend is an indication of increased function-level leakage with collusion. By contrast, FH-SMPC does not exhibit any divergence between all adversarial configurations, which validates the claim that encapsulation of functions and execution patterns that are uniform in all adversarial configurations are enough to encourage transcript-based inference to be suppressed even at or close to the reconstruction threshold. The latency of the marginal decrease in execution does not increase with adversarial strength.

4.11 Effect Size and Practical Significance

In addition to statistical significance, there is a need to determine whether the leaptrof improvements are truly significant. The decrease in transcript divergence attained by FH-SMPC is associated with a significant effect size, which means that the change is not only an incremental change but a significant change in leakage characteristics. In all the assessed conditions, Dtrans reduces by more than 85% compared to baseline SMPC, which is conveniently within the range of effects considered by applied security analysis as strong. This finding supports the finding that function hiding in FH-SMPC produces tangible security payoffs and not theoretical marginal payoffs.

4.12 Result Reproducibility and Validation Context

The entirety of results here is possible to reproduce the entire workflow of the experiment, along with the dataset specification and evaluation measures, as detailed in Section 3.8, 3.9, and 3.11. Experiments were repeated with the same conditions and random seeds were kept constant and results were released as averaged values of the same experiment across the runs. The post hoc parameter tuning was not done to support the proposed parameter.
Such overt connection between the methodology and the results will provide the empirical results presented here to be validated independently and to be reliably extended in further research.

4.13 Consolidated Result Interpretation

The combination of the quantitative comparison, statistical validation, ablation study, adversarial robustness analysis, and effect-size interpretation have offered a logical and holistic assessment of the proposed FH-SMPC framework. These findings indicate that it is possible to enhance at least the level of privacy at functionality without affecting correctness or scalability. More to the point, they assure that the security benefits that they have observed remain in place with realistic adversarial conditions and withstand protocol-level stress factors.

5 . Discussion

The findings in Section 4 show trends that go beyond the increase of the numerical values of the leakage measures or execution times. The fact that transcript divergence was significantly smaller than that incurred by FH-SMPC even when the adversarial strength is increased indicates that exposure to functions in classical SMPC is not a peripheral phenomenon but a structural leak. This finding substantiates the main hypothesis of this paper, according to which the computation logic as such is a sensitive resource and should be safeguarded alongside other private inputs. The success of encapsulating functions implies that to a large extent, the leakage does not come as a result of cryptographic weakness, it comes as a result of observable execution regularities based on the structure of functions. FH-SMPC essentially modifies the leakage surface in the novel way that does not just contract the size of the leakage surface.

FH-SMPC is a protocol in between expressive SMPC protocols that prioritize computational generality [1] and leakage-resilient primitives that prioritize input exposure [7]. Most of the previous methods largely assume that the role is a social one, and that leakage is an undesirable by-product of communication. Equally, functional encryption schemes and function-security models, attain strong function privacy, but outside the interactive SMPC context, which restricts their practical use [20], [30]. On the contrary, FH-SMPC incorporates directly into an SMPC workflow functional hiding without compromising interactivity and scalability. This integration is the reason why the proposed framework has better leakage suppression than classical SMPC and is also much more efficient than separate function-hiding cryptographic constructions.

On theoretical grounds, the results support the need to broaden the conventional definitions of SMPC security by including a specific mention of the leakage at the function level. A somewhat surprising, but significant finding is that patterns of uniform executions, in conjunction with the encapsulation of functions, can be disproportionately large privacy gains, as compared to their size in terms of computation [35-39]. This implies that protocol architecture can be an asset to future protocol design as cryptographic expertise [40-45].

However, in spite of the strengths, the proposed framework has limitations. Testing is done in controlled conditions on synthetic workloads, which though required to isolate leakage characteristics, does not necessarily reflect all the dynamics of distributed deployments on a large scale. Also, the existing implementation is not dealing with side channels at hardware level or denial-of-service attacks, which are not in the defined threat model. Further potential work might involve the use of structures of optimized encapsulation and benchmark of real-world applications as well as their integration with adaptive adversary models.

In a wider context, the work fits the new debate on transparency in algorithms, protection against intellectual property, and secure computing organizations. FH-SMPC addresses a conceptual difference between cryptographic privacy and system development by seeing functions as confidential objects. By so doing, it helps on a developing trend of holistic privacy forms that safeguard what is computed and not the source of data.

6. Conclusion and Future Scope

This paper dealt with a frequently ignored but progressively important problem in secure multiparty computation the danger of information leakage due to disclosure of computation logic. Although the conventional SMPC models offer good assurances on the confidentiality of inputs, they predominantly rely on the fact that the task under assessment is non-sensitive and publicly available. In the context of formal analysis and empirical assessment, this paper has shown that this assumption may result into non-trivial leakage, especially in an environment where computation logic codes proprietary strategies, decision policies, or sensitive analytical models. The paper has filled this gap by proposing the Function-Hiding Secure Multiparty Computation (FH-SMPC) framework which incorporates encapsulation of functions and homogenous execution patterns into the workflow of SMPC. The offered solution maintains accuracy and scalability and ensures a substantial decrease in transcript-level distinguishability. Theoretical security analysis and long experimental data validated that FH-SMPC has high-function indistinguishability and no prohibitive computational or communication rates. The use of quantitative comparisons to prove the results, statistical validation, and studies of ablation confirmed that the observable improvements can be explained by conscious architectural decisions and not by accidental phenomena. In addition to the direct contributions, this work shows a more general change in the conceptualization of privacy in collaborative computation. Guaranteeing the safety of inputs is no longer effective in hostile and competitive world; computation logic should be a first-class confidential asset. This study preconditions the development of more privacy-preserving computation models by proving the possibility of effective implementation of the concept of function hiding in the context of SMPC.

A few recommendations on the way to conduct future research will follow. It would be possible to test the scalability of the framework to large-scale, heterogeneous network environments and to test its performance with real-world workloads, which would help in further legitimizing its practicability. Future research on adaptive adversary models, combining adaptive adversary models with trusted execution environments, or resilience-enhancing hybrids between function hiding and differential privacy can result in further resilience. Lastly, making the formalization of function-level privacy as a standard security dimension an SMPC design choice may have an impact on future protocol design and benchmarking. To conclude, FH-SMPC is a move towards more comprehensive secure computational models - models that can ensure the integrity of not only data, but also of logic that manipulates it.

Acknowledgements
Not applicable.

Funding
This work was supported by the Deanship of Scientific Research, Vice Presidency for Graduate Studies and Scientific Research, King Faisal University, Saudi Arabia (Grant No. KFU260480).

Contributions
U.M; V.A; I.K; A.A; R.S; M.O: Conceptualization, Methodology, Software, Writing – Original Draft, Visualization, Project Administration. U.M: Data Curation, Writing – Review & Editing, Supervision.

Ethics declarations
This article does not contain any studies with human participants or animals performed by any of the authors.

Consent for publication
Not applicable.

Competing interests
The authors have no conflicts of interest to disclose.

References

[1]. S. R. H. Najarkolaei, M. M. Mojahedian, and M. R. Aref, "Beyond Yao's Millionaires: Secure Multi-Party Computation of Non-Polynomial Functions," Oct. 2024, doi: 10.48550/arxiv.2410.17000.
[2]. C. Wang et al., "TSS-ZKP: Enabling Blockchain Account Supervision Without Compromising User Identity," IEEE Internet of Things Journal, doi: 10.1109/jiot.2025.3577616.
[3]. M. Kumar and B. Mondal, "Quantum Secure Multiparty Computation based on Secure Summation and QKD," Jul. 2025, doi: 10.21203/rs.3.rs-6706796/v1.
[4]. S. Tarnopolsky, Zirui, S. Deng, V. Ramkumar, N. Raviv, and A. Cohen, "Individual Confidential Computing of Polynomials over Non-Uniform Information," Jan. 2025, doi: 10.48550/arxiv.2501.15645.
[5]. Aslam, W. Tariq, T. Waheed, K. Hamid, S. Rizwan, and A. Ahmed, "Enhancing Privacy and Security in Multi-Party Computation for Data Mining in Cryptographically Protected Environments," vol. 3, no. 8, pp. 510–531, Aug. 2025, doi: 10.63075/fxe5gg65.
[6]. X. Liu et al., "HE/MPC-Based Scheme for Secure Computing LCM/GCD and Its Application to Federated Learning," Symmetry, vol. 17, no. 7, p. 1151, Jul. 2025, doi: 10.3390/sym17071151.
[7]. C. Hazay, M. Venkitasubramaniam, and M. Weiss, "Protecting Distributed Primitives Against Leakage: Equivocal Secret Sharing and more," Journal of Cryptology, vol. 38, no. 1, Oct. 2024, doi: 10.1007/s00145-024-09524-3.
[8]. S. Begum and S. Sultana, "Anonymous data leakage-resilient in multi-receiver hybrid data encryption in public key system," vol. 21, no. 3 (1), pp. 1516–1522, Aug. 2025, doi: 10.62643/ijerst.v21.n3(1).pp1516-1522.
[9]. R. R. Cohen, J. Doerner, Y. Kondi, and A. Shelat, "Secure Multiparty Computation with Identifiable Abort via Vindicating Release," pp. 36–73, doi: 10.1007/978-3-031-68397-8_2.
[10]. Q. Deng, L. Chen, and Y. Mu, "Leakage Reduced Searchable Symmetric Encryption for Multi-keyword Queries," IEEE Transactions on Cloud Computing, doi: 10.1109/tcc.2025.3573378.

ISSN: 3079-5354

[11]. H. Chen, J. Li, and L. F. Zhang, "A Multi-Server Publicly Verifiable Computation Scheme with Context-Hiding Property," pp. 1860–1865, Jul. 2024, doi: 10.1109/isit57864.2024.10619256.
[12]. Q. Gao, D. Lu, D. Wen, S. Renqian, and X. Wei, "Multi-party verifiable arbitrated quantum signature scheme with information hiding function," Laser Physics Letters, vol. 22, no. 8, p. 085202, Jul. 2025, doi: 10.1088/1612-202x/adf395.
[13]. S. K. Dixit, A. Ramamoorthy, and K. Anusha, "Neural Cryptographic Protocols Using Secure Multi-Party Computation (SMPC) for Encrypted Data Processing in AI-Driven Security System," pp. 519–549, Mar. 2025, doi: 10.71443/9789349552029-17.
[14]. M. Thomas, L. Choudhary, Analysis of cloud security and performance for leakage of critical information using DROPS methodology, International Research Journal of Engineering and Technology (IRJET), vol. 7, no. 2, pp. 103–105, Feb. 2020. [Online]. Available: www.irjet.net
[15]. G. Zhao, W. Tan, and C. Peng, "An Attribute-Based Proxy Re-Encryption Scheme Supporting Revocable Access Control," Electronics, vol. 14, no. 15, p. 2988, Jul. 2025, doi: 10.3390/electronics14152988.
[16]. Q. Ye and B. Delaware, "Taypsi: Static Enforcement of Privacy Policies for Policy-Agnostic Oblivious Computation," Proceedings of the ACM on programming languages, vol. 8, no. OOPSLA1, pp. 1407–1436, Apr. 2024, doi: 10.1145/3649861.
[17]. N. K. Rathi, "Beyond Encryption: A Holistic Approach to Privacy-Preserving Query Processing in Modern Database Systems," Indian Scientific Journal Of Research In Engineering And Management, vol. 09, no. 06, pp. 1–9, Jun. 2025, doi: 10.55041/ijsrem49974.
[18]. A. Syed, "AI-powered genomic risk prediction and personalized drug recommendations using transformers," in Transformative Role of Transformer Models in Healthcare, © 2026, pp. 32, doi: 10.4018/979-8-3373-2038-0.ch010.
[19]. M. Xu et al., "Sectric: Towards Accurate, Privacy-Preserving and Efficient Triangle Counting," Proceedings of The Vldb Endowment, vol. 18, no. 10, pp. 3382–3395, Jun. 2025, doi: 10.14778/3748191.3748202.
[20]. Y. Li, J. Wei, F. Guo, Y. Xiang, and X. Chen, "Function-Hiding Multi-Client Inner-Product Functional Encryption Without Pairings for Large Space," IEEE Transactions on Dependable and Secure Computing, doi: 10.1109/tdsc.2025.3576164.
[21]. H. Choi, "PP-STAT: An Efficient Privacy-Preserving Statistical Analysis Framework using Homomorphic Encryption," Aug. 2025, doi: 10.1145/3746252.3761194.
[22]. V. R. Cadambe, H. Jeong, and F. P. Calmon, "Differentially Private Secure Multiplication: Hiding Information in the Rubble of Noise," pp. 2207–2212, Jun. 2023, doi: 10.1109/isit54713.2023.10206949.
[23]. B. Han et al., "PBFL: A Privacy-Preserving Blockchain-Based Federated Learning Framework With Homomorphic Encryption and Single Masking," IEEE Internet of Things Journal, p. 1, Jan. 2025, doi: 10.1109/jiot.2024.3524632.
[24]. Y. Chen, N. Ding, D. Gu, and Y. Bian, "Practical multi-party private set intersection cardinality and intersection-sum protocols under arbitrary collusion1," Journal of Computer Security, Apr. 2024, doi: 10.3233/jcs-230091.
[25]. M. Vijaya and L. S. Chakravarthy, "SECUREDGE: Privacy-Preserving Deduplication with Homomorphic Encryption for Multi-Tenant Cloud Systems," International Journal of Basic and Applied Sciences, vol. 14, no. 3, pp. 242–257, Jul. 2025, doi: 10.14419/6ekp0r85.
[26]. J. Zhang et al., "A Secure and Lightweight Multi-Party Private Intersection-Sum Scheme over a Symmetric Cryptosystem," Symmetry, vol. 15, no. 2, p. 319, Jan. 2023, doi: 10.3390/sym15020319.
[27]. F. Falzon and E. A. Markatou, "Re-visiting Authorized Private Set Intersection: A New Privacy-Preserving Variant and Two Protocols," IACR Cryptology ePrint Archive, vol. 2024, p. 1579, Jan. 2025, doi: 10.56553/popets-2025-0041.
[28]. Y. Luo, Y. Chen, T. Li, C. J. Tan, and H. Dou, "Cloud-SMPC: two-round multilinear maps secure multiparty computation based on LWE assumption," Jan. 2024, doi: 10.1186/s13677-023-00586-5.
[29]. H. Xia and M. Wang, "Efficient and Compact Full-Domain Functional Bootstrapping via Subring Folding," IACR transactions on cryptographic hardware and embedded systems, vol. 2025, no. 4, pp. 737–762, Sep. 2025, doi: 10.46586/tches.v2025.i4.737-762.
[30]. Y. Bai, X. Guang, and R. W. Yeung, "Secure Network Function Computation: Function-Security," pp. 1514–1519, Jul. 2024, doi: 10.1109/isit57864.2024.10619365.
[31]. X. Kang et al., "Privacy Set Intersection From Oblivious Polynomial Evaluation," Zhongguo kexue, Aug. 2025, doi: 10.1360/ssi-2025-0184.

[32]. S. Saeidian, G. Cervia, T. J. Oechtering, and M. Skoglund, “Rethinking Disclosure Prevention with Pointwise Maximal Leakage,” The journal of privacy and confidentiality, vol. 15, no. 1, Mar. 2025, doi: 10.29012/jpc.893.

[33]. Sweet, D. Darais, D. Heath, W. R. Harris, R. Estes, and M. Hicks, “Symphony: Expressive Secure Multiparty Computation with Coordination,” The art, science, and engineering of programming, vol. 7, no. 3, Feb. 2023, doi: 10.22152/programming-journal.org/2023/7/14.

[34]. H. S. Jaafar, A. A. Abed, and M. A. Al-Shareeda, “A Secure Industrial Internet of Things (IIoT) Framework for Real-Time PI Control and Cloud-Integrated Industrial Monitoring,” STAP Journal of Security Risk Management, vol. 2026, no. 1, pp. 77–86, 2026, doi: 10.63180/jsrm.thestap.2026.1.5.

[35]. Alsahaim S, Almaiah MA, Sulaiman RB. Security Threats in Mobile Phones: Challenges, Countermeasures, and the Importance of User Awareness. International Journal of Cybersecurity Engineering and Innovation. 2023 Nov 25;2023(1).

[36]. K. Audah Kareem and M. A. Al-Shareeda, “A Low-Complexity Li-Fi Communication Framework for Short-Range Text Transmission,” Jordanian Journal of Informatics and Computing, vol. 2026, no. 1, pp. 15–24, 2026, doi: 10.63180/jjic.thestap.2026.1.2.

[37]. H. S. Jaafar, A. A. Abed, and M. A. Al-Shareeda, “A Secure Industrial Internet of Things (IIoT) Framework for Real-Time PI Control and Cloud-Integrated Industrial Monitoring,” STAP Journal of Security Risk Management, vol. 2026, no. 1, pp. 77–86, 2026, doi: 10.63180/jsrm.thestap.2026.1.5.

[38]. Alghareeb MS, Almaiah M, Badr Y. Cyber Security Threats in Wireless LAN: A Literature Review. International Journal of Cybersecurity Engineering and Innovation. 2024 Dec 31;2024(1).

[39]. S. R. Addula, S. Norozpour, and M. Amin, “Risk Assessment for Identifying Threats, vulnerabilities and countermeasures in Cloud Computing,” Jordanian Journal of Informatics and Computing, vol. 2025, no. 1, pp. 38–48, 2025, doi: 10.63180/jjic.thestap.2025.1.5.

[40]. Alrajeh M, Almaiah M, Mamodiya U. Cyber Risk Analysis and Security Practices in Industrial Manufacturing: Empirical Evidence and Literature Insights. International Journal of Cybersecurity Engineering and Innovation. 2026 Jan 7;2026(1).

[41]. B. Konda, A. R. Yadulla, V. K. Kasula, M. Yenugula, and S. B. Rakki, “A Quantum-Resistant Privacy-Preserving Framework for Consortium Blockchains Using Blind Signatures, Hierarchical Fully Homomorphic Encryption, and Zero-Knowledge Proofs,” pp. 1–6, Feb. 2025, doi: 10.1109/satc65530.2025.11137073.

[42]. Bay, “Delegated multi-party private set intersections from extendable output functions,” PeerJ, vol. 11, p. e3141, Aug. 2025, doi: 10.7717/peerj-cs.3141.

[43]. Yassin A, Almaiah M. Cyber security risk assessment for determining threats and countermeasures for banking systems. International Journal of Cybersecurity Engineering and Innovation. 2026 Jan 9;2026(1).

[44]. Gruner, C.-A. Brust, and A. Zeller, “Finding Information Leaks with Information Flow Fuzzing - RCR Report,” ACM Transactions on Software Engineering and Methodology, Jan. 2025, doi: 10.1145/3711905.