# Dual-Cloud Multi-Secret Sharing Architecture for Privacy Preserving Persistent Computation

Yu-Chi Chen, *Member, IEEE,* Jhe-Kai Yang, Hsin-Chan Yen, and Pei-Wen Lin

***Abstract*—With the prevalence of artificial intelligence, people collect data through numerous sensors and use machine learning to create models for intelligent services. However, data privacy and massive data issues are raised with the proliferation of devices. Although secret sharing can be the solution to providing privacy and performing efficient computations (than homomorphic encryption-based) in the privacy domain, sharing large amounts of data may impose a considerable storage burden on resource-restricted devices. Therefore, our goal is to reduce the shares that participants need to hold and construct secure computation protocols that form privacy-preserving data utilization with multi-secret sharing. In this paper, we study the mechanism of persistent computation with multi-secret sharing for privacy protection, which avoids the requirements for participants to store extra information during secure computation. We present the mask techniques to convert secrets into protected data and upload them separately to non-colluding dual-cloud servers through multi-secret sharing. Then, we can locally evaluate the operation result shares by revealing a part of the protected data, where the true secret consistently remains secure as the other part stays in a shared state. Eventually, we outsource a dataset to the cloud and build a privacy-preserving multi-party $k$NN classification based on our scheme, and provide some experiments to demonstrate the feasibility and usability of storage size.**

***Index Terms*—Dual-cloud, Privacy-preserving computation, Multi-secret sharing.**

## I. Introduction

ARTIFICIAL intelligence (AI) is one of the critical developments in the future, which can solve problems through training and inferring. Many companies began integrating AI into their products and services, such as face recognition [1] and decision recommendation [2]. Generally, we will apply machine learning algorithms to data and make an induction analysis of instances into a model. After that, we can infer optimal decisions through statistical approximation situations in the model. Since considering more relevant data from different sources is necessary for training a more robust model, the opportunity to share and integrate data among multiple parties increases as we intend to form better intelligent services. However, data becomes an important asset. Although there are some studies [3]–[5] focusing on securely collecting data, people may worry that their sensitive data will be leakage due to the outflow of integrated data. Therefore, people still want to supervise their shared data while others usage.

Y.-C. Chen, H.-C. Yen, and P.-W. Lin are with the Department of Computer Science and Information Engineering, National Taipei University of Technology, Taipei, Taiwan
(e-mail: wycchen@ieee.org (Y.-C. Chen))
J.-K. Yang was with the Department of Computer Science and Engineering, Yuan Ze University, Taoyuan, Taiwan
Manuscript received September 13, 2023; revised March 17, 2024.

Shamir's secret sharing [6] is a cryptographic primitive that can make a secret into multiple shares and recover the secret with enough shares. A $t$-out-of-$n$ secret sharing protect scheme will distribute shares among $n$ participants and retrieve a piece of confidential information only if sufficient $t$ participants cooperate. Since the recovery requires participant collaboration, secret sharing has the advantage of overseeing the employment of shared resources. However, a share can only handle a single secret in traditional setting, which means each participant needs to hold massive shares for the entire system. Indeed, we cannot always expect a perfect environment that has unbounded resources with unlimited computing, storage, or communication power. As an extension of secret sharing, multi-secret sharing (MSS) can imply multiple secrets in a sharing process.

In traditional multi-secret sharing, a share can maintain several secrets at once. While some research [7] points out a bound of $d$ secrets ($d \leq t < n$), a novel MSS [8] scheme with public information bypasses the barrier and achieves client-side cost optimization. We denote the multi-secret sharing with public information as MSS since we do not consider the traditional ones in this paper. Along with the new type of MSS (as shown in Figure 1), we will treat the client-friendly approaches as a lightweight solution to achieve cooperative computing systems on resource-restricted devices.

Table I: Resource overhead on fully homomorphic encryption, secret sharing, and multi-secret sharing.

| Scheme | Client | | Server | |
|---|---|---|---|---|
| | Preparation | Running | Preparation | Running |
| FHE | High | High | - | High |
| SS | middle | middle | - | Low |
| MSS | Low | Low | Low | middle |

Unfortunately, multi-secret sharing cannot directly provide many functionalities like fully homomorphic encryption (FHE). Secure computation is a technique that allows performing calculations over ciphertexts and is often a functional need in secure systems. FHE seems fitting but requires more computation cost. Table I shows the resource overhead over these schemes. Although secret sharing also provides the property of homomorphic addition that does not rely on complex calculations, it becomes troublesome while multiplying shares to reconstruct the product.

Secret sharing derives from polynomials, which support share addition that implies the sum of secrets. The secret multiplication is equivalent to a polynomial multiplying a polynomial, which makes the polynomial degree higher and raises the threshold. Some works use polynomial reduction [9] to solve the lacking share problem, but it becomes difficult in MSS interactions. We found that Shingu et al. [10] gave a clever idea to deform two polynomials into a polynomial and a scalar value during secure multiplication. This scheme uses a random number as a mask to convert the secret into a hidden secret called concealed secret, then distributes the hidden secret by secret sharing. After that, we can securely reveal the concealed secret and multiply with shares without changing the polynomial degree. Additionally, their construction also allows simple homomorphic addition calculation.

Nevertheless, secure computation will generate a new set of shares representing the calculation result. While persistent calculation occurs, we may do more computations above some result shares, which makes massive temporary shares a storage burden. These additional shares go against the concept that a user maintains a single share in cloud assistance multi-secret sharing (CAMSS). Therefore, avoiding participants from holding more shares during computation is a significant challenge without doubt. Eventually, we overcome this and then propose a privacy-preserving computing architecture based on multi-secret sharing and design a series of protocols that support persistent computing for resource-restricted devices. At last, we return origin and realize secure machine learning with our system. In the experiment, we demonstrate the feasibility and usability of our work under vast ciphertext[1] computations through privacy-preserving $k$-nearest neighbor ($k$NN) classification.

## A. Contributions

In this paper, we propose a privacy-preserving persistent computation architecture based on dual-cloud and multi-secret sharing (DCMSS). The main contributions of this paper are roughly composed of *architecture and building blocks*, *persistent computation*, and *applications*. They are summarized as follows.

1) We build a secure computation architecture that can supervise and calculate simply for the shared secrets through DCMSS. Particularly, the clients can be resource-restricted devices with only fewer storage. Comparing to homomorphic encryption-based solution, it is more efficient on time and space complexity. Comparing to traditional secret sharing-based solution, it is more efficient on space complexity.
2) By the architecture and building blocks, we further design a series of DCMSS protocols that can support succint persistent computation. In fact, if we do not have the presented protocols, the number of shares increases as long as we perform operations. It is very significant to overcome this challenge, and these protocols offer a manner without holding more shares but can complete the computation tasks.
3) We test the feasibility of our work and show the usability with vast operations in privacy-preserving $k$-nearest neighbor classification. Moreover, in the experiment, we also demonstrate model training maintains high accuracy for several real-world datasets under our scheme. Also, some trivial techniques are discussed to help to realize privacy preserving applications.

[1]In this paper, we abuse the unified word, *ciphertext*, to denote ciphertext in encryption or share in sharing. Also, we say the privacy domain means the encrypted domain or secret share domain.

## B. Organization

The remainder of this paper organizes as follows. In Section II, we review the related works of secret sharing and secure computation. Section III introduces some preliminaries used in this work. Section IV mentions our system model and design goals. Section V describes the structure and protocol we propose. Section VI will talk about the application of our work. Section VII gives the security analysis. Section VIII shows the experimental results. Finally, the conclusions of this paper give in Section IX.

## II. Related Works

For dealing with secure multiparty computation on IoT or mobile devices, we will review existing research about secure computation. Meanwhile, we will comprehensively investigate secret sharing and sort out previous research involving multi-secret sharing techniques in the survey.

## A. Secure Computation

Secure computing is the basis for providing secure service and has been extensively discussed in various literature. To avoid data leaks from intermediate messages, fully homomorphic encryption (FHE) and secure multi-party computation (SMC) are usual methods for privacy-preserving computing. Over recent years, the demand for secure computation has garnered widespread attention from the exploitation of big data and outsourced computation with privacy-preserving. This makes a deeper dive into the underlying tools more meaningful.

## B. Fully Homomorphic encryption

Fully homomorphic encryption [11] is a cryptographic scheme that enables homomorphic operations[2] on encrypted data without decryption. Although FHE provides strong privacy with noise, its computational efficiency is low and requires more resources. Owing to its low efficiency, Koseki et al. [12] build a lightweight strategy by stochastic computing [13], which mitigates the computational costs from FHE complex systems. To deal with the noise problem, Cheon et al. [14] proposed a rescaling technique for approximate arithmetic homomorphic encryption, which makes errors grow linearly instead of error growing exponentially on the number of multiplications as the conventional FHE. Recently, the

[2]Note that there exist somewhat homomorphic encryption schemes. For example, RSA, ElGamal, and Paillier encryption schemes only support one operation. However, they can be used to realize SMC, but induce more communication overhead in protocol design.

Cheon-Kim-Kim-Song (CKKS) scheme [14] has become one of the most widely used FHE schemes.

Among these research directions, substantial works are concerned with the performance and cost of computing over the confidential domain. Jiang et al. [15] discover general FHE schemes suffer from significant inefficiency in both computation and communication. Hesamifard et al. [16] point out that SMC protocols based on secret sharing will have higher communication overhead than FHE protocols, which especially grows significantly as neural networks become more complex. Despite these overhead drawback, we remain more optimistic about the performance of secret sharing-based SMC as long as we can have the other benefits.

### C. Secret Sharing

Secret sharing is an essential primitive that does not need complicated computations, which is also a well-known unconditional security method. Without hardness assumptions, it has become a lightweight strategy for secure multi-party computations on IoT or mobile devices. Moreover, it is also expected as an approach to resist quantum attacks. Abundant applications of secret sharing have thrived in the research [17]–[19]. Kaur et al. [17] proposed an interesting 2-line secret sharing approach for generating diverse pseudo-biometric identities for accessing different applications from a single user-specific token. Chor et al. [18] started the verifiable secret sharing scheme, which solves the Byzantine faults problems through simultaneous broadcast with fault-tolerant distributed computing. Then, Stadler [19] presented publicly verifiable secret sharing, which provides a verification mechanism that allows everyone to verify the correctness of the shares, not only the participants.

As an option to improve cryptographic performance, secret sharing is also not invincible. Ren et al. [20] devoted to reducing the complex communication in neural networks. Shivhare et al. [21] provide secret sharing as a suitable method for the constrained scenario, which requires a low cost on the IoT devices side. Nevertheless, they mention that the memory requirements may become a limitation in implementation aspects. Yang et al. [22] speed up the performance by partitioning data into groups at the preprocessing phase. Since communication is not a critical problem in many real-world applications, we pay more attention to computational capability and memory cost issues, whereas $k$NN may be a proper experimental subject.

### D. Extension for Secret Sharing

Apart from the above secret sharing-based studies, Attasena et al. [23] survey the security for secret sharing and cloud data. From this survey, we not only can glance at various secret sharing but also peek at some multi-secret sharing technologies that deal with the drawback of multiplying initial data volume by the number of participants. After that, we feature two papers [24], [25] as multi-secret sharing to display an extension for secret sharing. Nag et al. [24] gave a multi-secret sharing scheme that hides multiple secrets at once and distributes them on $n$ clouds. However, the secrets are all packed together and lose the ability to modify particular ones. Tang [25] proposed an exciting scheme to reduce the storage overhead at clients, thus maximizing the advantages that cloud-based technology can offer to the resource-restricted IoT system. Unfortunately, this work only supports multiplication operations. In the later sections, we intend to adopt multi-secret sharing into resource-restricted scenarios while exhibiting better arithmetic flexibility.

## III. Preliminaries

### A. Multi-Secret Sharing

Multi-secret sharing allows users to hold the same share for reconstructing multiple secrets. In our construction, we focus on the MSS scheme proposed in [8]. Firstly, we decide the participant shares $S$, which are some shares for each secret. Then, generate the other shares as public information $pub$ from secrets and fixed shares. Finally, we can recover the secret $K$ with at least $t$ participant shares and the public shares. We call this optimal client-side solution of placing other information on the cloud and establishing a single share for each user as a cloud assistance multi-secret sharing (CAMSS). Note that we can also represent $[K]$ as the participant shares and public shares of the secret $K$ in protocols.

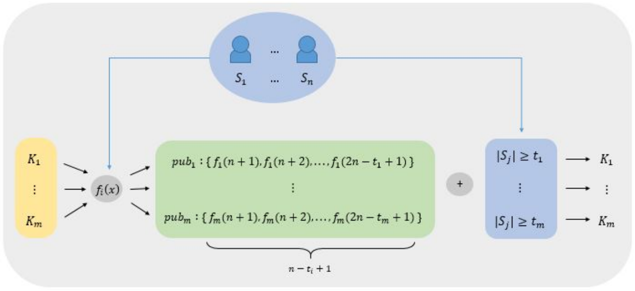

$S_1$ ... $S_n$
$K_1$
$K_m$
$f_i(x)$
$pub_1 : \{f_1(n+1), f_1(n+2), \dots, f_1(2n-t_1+1)\}$
$pub_m : \{f_m(n+1), f_m(n+2), \dots, f_m(2n-t_m+1)\}$
$n - t_i + 1$
+
$|S_j| \geq t_1$
$|S_j| \geq t_m$
$K_1$
$K_m$

Figure 1: MSS mechanism for sharing and recovery.

Figure 1 shows the sharing and recovering process in the MSS scheme, where there are $m$ secrets to be shared among $n$ participants $(t \leq n)$. Then MSS scheme needs to meet the following terms:

- According to the public shares $pub_i$, any $t_i$ or more than $t_i$ users can recover the secret $K_i$.
- If there is no public share, we cannot obtain any information even if all users participate.
- The secret should not be revealed while there are less than $t_i$ participants.

In the sharing process, we get $n+1$ points $(0, K_i)$, $(j, S_j)$ for each secret $K_i$ by participant share $S_j$, where $j = \{1, 2, ..., n\}$. Then, we consist $f_i(x)$ of the following formula:

$$f_i(x) = K_i \prod_{j=1}^{n}(1 - \frac{x}{j}) + \sum_{j=1}^{n} S_j \prod_{l=0, l \neq j}^{n} \frac{x-l}{j-l}$$

, where $i = \{1, 2, ..., m\}$. After that, we can get the public shares $pub_i = \{f_i(n+1), f_i(n+2), ..., f_i(2n - t_i + 1)\}$. And finally, any $t_i$ participant shares and $n - t_i + 1$ public shares in $pub_i$ can recover secret $K_i$ by reconstructing a $n$+1-out-of-$n$+1 secret sharing under a prime modulo $p$.

We can present the MSS scheme as follows:

- Share function creates public shares from the secret and the participant shares. $Share(K_i, S) \rightarrow pub_i$.
- Recover function will take enough participant shares and the public shares to rebuild the secret. $Recover(pub_i, S) \rightarrow K_i$.

## B. Beaver Triples

Beaver et al. [26] introduce an approach to deal with secure multi-party multiplication issue in secret sharing, called multiplication triples (also often called Beaver triples). Let $[z]$ denote the share of secret $z = x \cdot y$, where the share of secret $x$ and $y$ are represented as $[x]$ and $[y]$. We can generate Beaver triples $([a], [b], [ab])$ to make $[\alpha] = [x] - [a]$ and $[\beta] = [y] - [b]$. Then, we recover $\alpha$ and $\beta$, which are secure for secrets since we distribute $a$ and $b$ as shares in each party. Lastly, we can locally computes $[z] = \alpha \cdot \beta + \alpha \cdot [b] + \beta \cdot [a] + [ab]$.
**Correctness:** The multiplication without degree increment supports local computation via scalar multiplication and share addition. We demonstrate the correctness below. Here we define $R^*$ as a general secret sharing recover function.

$$\begin{aligned} z &= R^*([z]) \\ &= R^*(\alpha \cdot \beta + \alpha \cdot [b] + \beta \cdot [a] + [ab]) \\ &= \alpha\beta + \alpha \cdot R^*([b]) + \beta \cdot R^*([a]) + R^*([ab]) \\ &= (x-a)(y-b) + (x-a) \cdot b + (y-b) \cdot a + ab \\ &= xy - ay - bx + ab + bx - ab + ay - ab + ab \\ &= xy \end{aligned}$$

## C. $k$-nearest Neighbor Algorithm

The $k$-nearest neighbor ($k$NN) algorithm is a case-based learning algorithm that computes the distances between the query data and the training instances, then determines the result by finding the $k$ nearest labels with distance comparison.

$k$-nearest neighbor algorithm:

1. Calculate the distances from the test instance to the instances in the training set.
2. Compare the distances and get the labels of the $k$-th nearest neighbors.
3. Set the test label as the major category among the nearest data.

Although $k$NN is simple and effective, memory requirements and computationally expensive are disadvantages [27] that cannot be ignored, which become significant challenges for privacy-preserving computing capabilities and storage on large datasets.

## IV. SYSTEM FRAMEWORK

### A. System Model

Figure 2 briefly presents the main flow of our system. Our construction contains four types of entities, e.g., a dealer, dual-cloud servers, multiple participants, and a randomness generator. Moreover, we define two record lists named randomness and operation in our system.

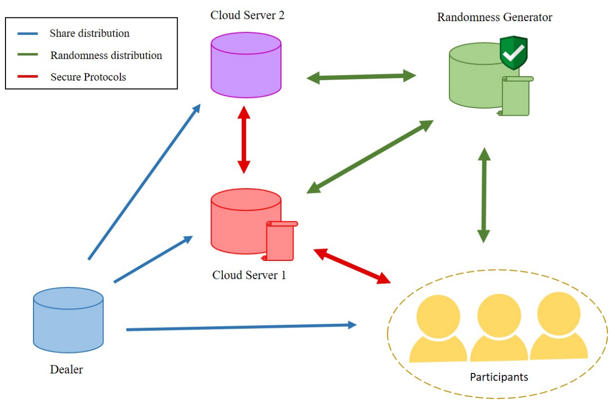


Figure 2: System model.

1) **Dealer**: The dealer is a trusted entity that collects data from multiple participants to form a database. Whenever we decide to build a service, the database will give a dataset $D$ that integrates instances from various participants to train a more robust model. Considering the cost and the limited computing power, we cannot provide the service simply through the dealer. So we introduce cloud computing for training models and providing services. However, the dealer is suspicious about the security of the cloud. Therefore, the dealer returns the management capabilities to all participants, as the dataset may contain sensitive data. In addition, the dealer will mask the data before outsourcing the dataset to the cloud. In our system, the dealer is responsible for system initialization which includes collecting and integrating data, distributing participant shares, masking the dataset, and outsourcing the public shares. After that, the dealer can go offline since we no longer need interaction with it in protocols.
2) **Dual-cloud servers**: The two non-colluding cloud servers are semi-honest entities, where $CS_1$ holds the public shares of the mask and $CS_2$ holds the public shares from the masked data. They both have abundant storage space and powerful computational ability, which support participants performing secure computations under the MSS system. We protect the privacy of raw data by preventing the simultaneous recovery of related secrets between the two servers. Furthermore, we build an operation record OpRec at $CS_1$, which saves a list of parameters for rebuilding the MSS operations. The OpRec allows more secure computations above the recorded operations, which makes temporary shares unnecessary stored in cloud servers and participants during persistent secure computation.
3) **Participants**: The participant $P$ is a data provider who holds a single share to maintain secrets on dual-cloud servers. They are also the service user who launches a query $Q$ to get the service from the outsourced dataset in our system. Since the participants can access the secret by satisfying the threshold, there must be enough collaborators to agree with the cooperative computation for completing the service. Although we do not initially share the query in the cloud, we can securely upload the

query data through the randomness generator and obtain the result from privacy-preserving computations with the servers.

4) **Randomness Generator**: The randomness generator $RG$ is a trusted third party that provides randomization parameters by converting the scalar multiplication whenever share usage. We can refresh the shares with different randomness parameters generated from the same number, which randomizes the mask shares in $CS_1$ and the share of the masked data in $CS_2$, respectively. Besides, we can also take advantage of this computing manner to transform the query data for the participant in our system. Similarly, we build a randomness record $\mathsf{RandRec}$ in $RG$, which holds the parameters for backtracking the randomizing shares and supports privacy-preserving during persistent computation.

**Remark:** We have defined two record lists to reduce the storage burden of temporary shares in persistent computation. Besides, we denote two statements for record operation as follows.

- Append statement adds a new line of parameters at the end of the record list.
- Delete statement clears the record list after the temporary data is no longer needed.

## *B. Threat Model*

In this paper, we assume that only the dealer and the randomness generator are trustable, which generate shares honestly and distribute them securely to all parties. On the other hand, we consider the participants and the dual-cloud servers to be semi-trusted, where they perform interactive protocols curious-but-honestly. That means they will faithfully follow the protocol for getting the correct results and try to infer the information held by others from the intermediate messages. Besides, the dual-cloud servers that do not collude are valid and widely used in cryptography works [28]. Using cloud servers belonging to two different companies with high social reputations can prevent them from colluding with each other since they would not take the risk of losing their market credibility.

Therefore, we introduce an adversary $\mathcal{A}$, whose goal is to gain unknown shares for higher access and leak the privacy of the dataset and query data. The adversary $\mathcal{A}$ can eavesdrop on the communication between the cloud servers and the participants to obtain the encrypted data and also get the assistance of the colluded parties. Although the dual-cloud servers collusion is restricted, the adversary $\mathcal{A}$ still can work together with a single cloud server and several participants in the worst case. Note that a powerful threat model for two cloud servers colluding with each other is beyond our discussion scope.

## *C. Design Goals*

Based on the system model and threat model, the multi-secret sharing computation architecture proposed in this paper achieves the following design goals.

1) **Privacy protection**: In our system, the dataset is a data collection from multiple participants, in which the secret must recover by at least $t$ participants. In other words, the computation should also reach a threshold in order to access and calculate the ciphertexts. Moreover, the non-collusive dual-cloud servers should learn no information and constitute a perfect cooperative pattern for multiple parties.
2) **Low storage**: Considering that the participant may use a resource-restricted device, we enable the participant to manage multiple secrets with the same share in multi-secret sharing. That is, the participant share does not grow with the size of the dataset, which gives more opportunities to apply in restricted situations, such as IoT devices, edge computing (autonomous vehicle driving system), or low Earth orbit satellite communication networks (Starlink).
3) **Persistency**: Computations based on secret sharing often generate new shares for the calculation results. However, this violates the low storage advantage of multi-secret sharing. Therefore, we expect to build secure protocols that would not increase shares during computations. Then extend them to a persistency calculation that can perform more ciphertext calculations on results without producing new shares. Note that we claim the computing manner is "persistent" as the computation is traceable back to the initial shares, which supports maintaining a single share for everyone under cloud assistance multi-secret sharing.
4) **Correctness**: The secure protocols we designed must be able to obtain correct results through ciphertext calculations. In the following experiment, we will compare the classification results obtained by the ordinary $k$NN algorithm and the privacy-preserving $k$NN built by our scheme. As long as the accuracy is close, we can confirm that it is a possible privacy-preserving solution for applications whose calculation amount is under $k$NN massive computations.

## V. Proposed Scheme

In this section, we first describe the mechanism of the distribution and reconstruction. Then, we present secure protocols as basic operations above multi-secret sharing. Table II are some notations we have in our scheme.

## *A. Distribution and Reconstruction*

Figure 3 shows an outsourced format and protected recovery of each secret. In the distribution phase, we randomly select a mask $\alpha$ for each data $x$ and obtain the masked data $\alpha \cdot x$. Then, we can outsource the protected data through MSS that involves the participant shares. That is, $CS_1$ holds the mask part as $[\alpha]_{pub}$, and $CS_2$ stores the masked data as $[\alpha x]_{pub}$. In the reconstruction phase, we pick a random number $r$ to refresh the shares of the protected data, where the mask and the masked data change but still maintain the same secret. Thus, we can protect the recovery in different rounds from exposing the information of shares due to the components previously disclosed on the servers. Finally, we can reconstruct

Table II: Notations.

| Notation | Description |
|---|---|
| $n$ | The number of participants. |
| $t$ | The threshold of the secret. |
| $\langle x\rangle = ([\alpha], [\alpha x])$ | The protected data of secret $x$. |
| $S_i(= [\alpha]_i = [\alpha x]_i)$ | A share of participant $P_i$ in MSS that can recover any protected data in a cloud server. |
| $[\alpha]_{pub}$ | A set of public shares in $CS_1$ that stands for the mask $\alpha$. |
| $[\alpha x]_{pub}$ | A set of public shares in $CS_2$ that stands for the masked data $\alpha x$. |
| $Share_{(S)}(x, t) = ([\alpha]_{pub}, [\alpha x]_{pub})$ | Distributing public shares above the participant shares $S$, where secret $x$ split as the protected data. |
| $Recover_{(t)}(S, [\alpha x]_{pub}) = \alpha x$ | Recovering the value of protected data from public shares and at least $t$ participant shares. |
| $\langle x + y\rangle$ | The MSSAdd computes the addition of protected data $\langle x\rangle$ and $\langle y\rangle$, where $x$ and $y$ are secrets in MSS. |
| $\langle x * y\rangle$ | The MSSMul computes the multiplication of protected data $\langle x\rangle$ and $\langle y\rangle$, where $x$ and $y$ are secrets in MSS. |

the secret through components on $CS_1$ and $CS_2$ at the same round. Note that for MSS consisting of secret sharing under a prime modulo $p$, all secrets should be less than $p$, including the mask, the masked data, and the refreshed ones.

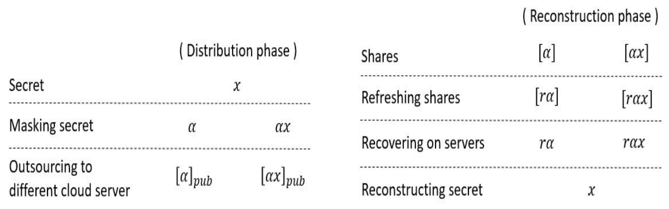
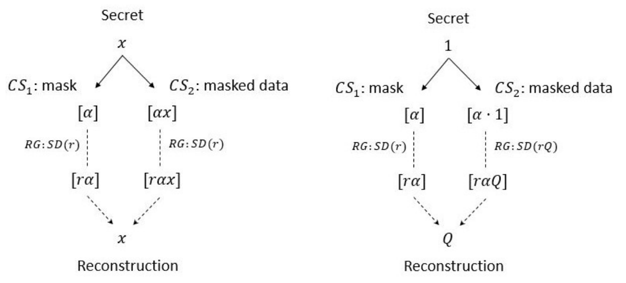


Figure 3: An example for distribution and reconstruction.



(a) Refreshing shares with a random number $r$.

(b) Uploading data $Q$ with a random mask $r$.

Figure 4: Scalar decentralized multiplication.

## B. Scalar Decentralized Multiplication

In our construction, we rely on the randomness generator to multiply the scalar into the share. While secret sharing already has the property of scalar multiplication, revealing the scalar value is unacceptable in MSS. In other words, the recovery of the multiplication result would expose the original secret by knowing the constant value during the local computation, where each party directly multiplies the constant value $c$ by their share $[x]$ to form a result share $[cx]$. Therefore, we design Protocol 1 to securely refresh the shares by decentralizing a scalar multiplication.

In this protocol, the randomness generator converts the multiplier into the form of secret sharing and applies Beaver triples for efficient share multiplication among multiple parties. Since secret sharing can make different shares out of the same number, we can securely refresh the shares by multiplying a random number into the mask and the masked data. Additionally, $RG$ will record the sending parameters $([y], [a], [b], [c])$ so that we can recall the previous refreshed shares through RandRec to form persistent computations.

**Protocol 1** Scalar Decentralized Multiplication (SD)

```
$RG$        **Input**: multiplier $y$, RandRec
$CS$        **Input**: $[x]_{pub}$
$P_i$       **Input**: $S_i(= [x]_i)$
$RG$        **Output**: RandRec$'$
$CS, P_i$   **Output**: $[xy]_i$
1. $RG$ randomly select $a, b$ and computes $c = ab$.
2. $RG$ generates $2n - t + 1$ shares $[y], [a], [b], [c]$.
3. $RG$ adds the parameters into the randomness record
   RandRec.$append([y], [a], [b], [c]) \rightarrow$ RandRec$'$.
4. $RG$ sends $[y]_i, [a]_i, [b]_i, [c]_i$ to $P_i$ and $CS$, respectively.
5. $P_i$ send $[x]_i - [a]_i$ and $[y]_i - [b]_i$ to $CS$.
6. $CS$ obtain $\varepsilon = x - a$ and $\rho = y - b$, and send them to $P_i$.
7. Finally, $P_i$ and $CS$ can locally computes
   $[xy]_i = \rho([x]_i - [a]_i) + \varepsilon[b]_i + \rho[a]_i + [c]_i$.
```

Moreover, we can modify the reconstructed secret by multiplying not equal numbers on the protected data. Whenever the mask increases, the reconstructed secret gets divided more. On the contrary, the reconstructed secret can multiply a number as large as the scalar multiplication of the masked data, which can become an alternative for share generation after the system initialization. Figure 4 illustrates an example of refreshing shares and uploading data.

## C. Multi-Secret Sharing Multiplication

Different from the SD protocol, which only performs scalar multiplication of the mask or the mask data. Protocol 2 can construct the product share of two secrets through the protected data. At the beginning of the MSS multiplication,

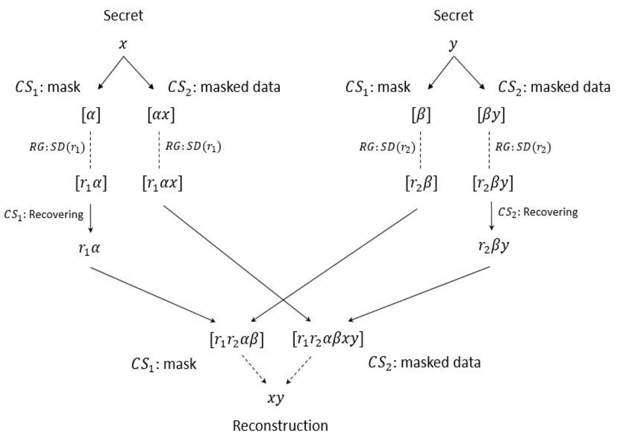


Figure 5: Multi-secret sharing multiplication.

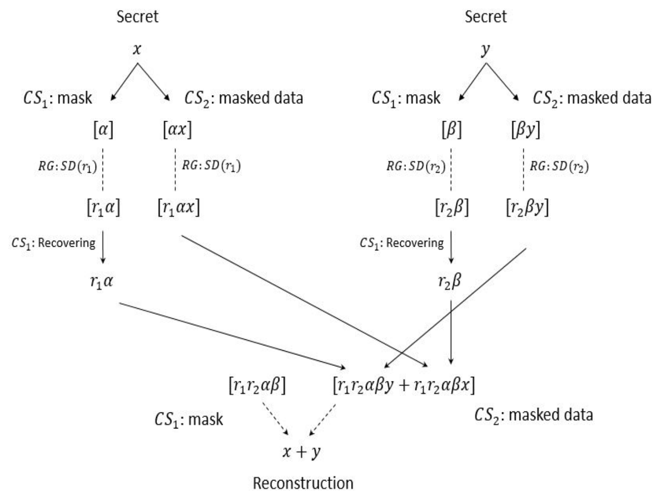


Figure 6: Multi-secret sharing addition.

$RG$ refreshes the protected data of both $x$ and $y$. Interestingly, even though the participants share $S_i(= [\alpha]_i = [\alpha x]_i)$ and multiplier $r_1$ are the same in Protocol 1 for the mask and the mask data, $[r_1\alpha]_i$ and $[r_1\alpha x]_i$ are distinct since the multiplier $r_1$ splits into different shares $[r_1]$ each time. That is, the refreshed participant share can only use for specific objects. As $[r_1\alpha]_i \neq [r_1\alpha x]_i$ for the same scalar $r_1$, $CS_2$ could recover nothing with $[r_1\alpha x]_{pub}$ even if $CS_2$ eavesdrop on the message $[r_1\alpha]_i$ that the participant passed to $CS_1$.

After that, we recover $r_1\alpha$ in $CS_1$ and $r_2\beta y$ in $CS_2$. The multiplication of two polynomials on secret sharing is deformed into a scalar and a polynomial so that the degree of the polynomial is not changed and the recovery can maintain the equivalent threshold for the calculation result. Figure 5 shows the calculation process of MSSMul. OpRec only needs to store two recovery values for rebuilding the product shares, and we can perform more MSS calculations above these shares.

**Protocol 2** Multi-Secret Sharing Multiplication (MSSMul)

| | |
|---|---|
| $CS_1$ | **Input**: $[\alpha]_{pub}$, $[\beta]_{pub}$, OpRec |
| $CS_2$ | **Input**: $[\alpha x]_{pub}$, $[\beta y]_{pub}$ |
| $P_i$ | **Input**: $S_i(= [\alpha]_i = [\beta]_i = [\alpha x]_i = [\beta y]_i)$ |
| $CS_1$ | **Output**: $[r_1r_2\alpha\beta]_{pub}$, OpRec$'$ |
| $CS_2$ | **Output**: $[r_1r_2\alpha\beta xy]_{pub}$ |
| $P_i$ | **Output**: $[r_1r_2\alpha\beta]_i$, $[r_1r_2\alpha\beta xy]_i$ |

1. $RG$ randomly selects two random multipliers $r_1$ and $r_2$ for $[\alpha], [\alpha x]$ and $[\beta], [\beta y]$ in Protocol 1.
2. $P_i$ sends $[r_1\alpha]_i$ to $CS_1$ then obtains $r_1\alpha$.
3. $P_i$ sends $[r_2\beta y]_i$ to $CS_2$ then obtains $r_2\beta y$.
4. $CS_1, CS_2$ adds the parameters to the operation record OpRec.$append(*, r_1\alpha, r_2\beta y) \rightarrow$ OpRec$'$.
5. Finally, $CS_1, CS_2, P_i$ receives $r_1\alpha$ and $r_2\beta y$. Then $CS_1$ computes $r_1\alpha[r_2\beta]_{pub}$, $CS_2$ computes $r_2\beta y[r_1\alpha x]_{pub}$, and $P_i$ locally computes $r_1\alpha[r_2\beta]_i, r_2\beta y[r_1\alpha x]_i$.

### *D. Multi-Secret Sharing Addition*

The linear relationship of secret sharing has the characteristic of summing secrets through shares addition. However, this could only help adjust a part of the protected data. Since the additive on the masks or the masked data does not directly mean the sum for the original secrets. We design Protocol 3 to build secret addition based on the masks and masked data.

Similarly, MSSAdd begins by refreshing the protected data of $x$ and $y$. Then, $CS_1$ recovers the masks $r_1\alpha$ and $r_2\beta$. After that, we multiply the mask data by the other mask value so the protected data can transform into the same mask $r_1r_2\alpha\beta$, and the sum of the masked data can imply the secret addition. Although we expose the value of the new mask, we compute $[r_1r_2\alpha\beta x]+[r_1r_2\alpha\beta y]$ without recovering $[r_1r_2\alpha\beta x]$ and $[r_1r_2\alpha\beta y]$. Thus, we do not leak the original secret while generating the summation shares, where the feature of available but invisible to the masked data also reflects in secret.

**Protocol 3** Multi-Secret Sharing Addition (MSSAdd)

| | |
|---|---|
| $CS_1$ | **Input**: $[\alpha]_{pub}$, $[\beta]_{pub}$, OpRec |
| $CS_2$ | **Input**: $[\alpha x]_{pub}$, $[\beta y]_{pub}$ |
| $P_i$ | **Input**: $S_i(= [\alpha]_i = [\beta]_i = [\alpha x]_i = [\beta y]_i)$ |
| $CS_1$ | **Output**: $[r_1r_2\alpha\beta]_{pub}$, OpRec$'$ |
| $CS_2$ | **Output**: $[r_1r_2\alpha\beta x + r_1r_2\alpha\beta y]_{pub}$ |
| $P_i$ | **Output**: $[r_1r_2\alpha\beta]_i$, $[r_1r_2\alpha\beta x + r_1r_2\alpha\beta y]_i$ |

1. $RG$ randomly selects two random multipliers $r_1$ and $r_2$ for $[\alpha], [\alpha x]$ and $[\beta], [\beta y]$ in Protocol 1.
2. $P_i$ sends $[r_1\alpha]_i$ to $CS_1$ then obtain $r_1\alpha$.
3. $P_i$ sends $[r_2\beta]_i$ to $CS_1$ then obtain $r_2\beta$.
4. $CS_1$ adds the parameters into the operation record OpRec.$append(+, r_1\alpha, r_2\beta) \rightarrow$ OpRec$'$.
5. Finally, $CS_1, CS_2, P_i$ receives $r_1\alpha$ and $r_2\beta$. Then $CS_1$ computes $r_2\beta[r_1\alpha]_{pub}$, $CS_2$ computes $r_2\beta[r_1\alpha x]_{pub} + r_1\alpha[r_2\beta y]_{pub}$, and $P_i$ locally computes $r_2\beta[r_1\alpha]_i, r_2\beta[r_1\alpha x]_i + r_1\alpha[r_2\beta y]_i$.

Figure 6 shows the calculation process of MSSAdd. OpRec provides the two masks to each party so that they can compute the summation share locally and do more MSS calculations with the result.

### *E. Secure Comparison*

While refreshing protected data is helpful for computational security, the secret remains intact from the mask and the masked data. Considering the comparison may leak the difference as soon as revealing the relationship between secrets, we should add randomness to them throughout the secure comparison.

Protocol 4 applies the above MSS operations to construct the modification on the standard value $\ell$ and the comparison value. For simplicity, we denote $\langle x\rangle = ([\alpha], [\alpha x])$ as the shares that offer by multiple parties, which is associated with secret $x$ for collaboratively computing MSS operations. Additionally, we represent MSSMul as $\langle \cdot * \cdot\rangle$ and MSSAdd as $\langle \cdot + \cdot\rangle$, which cooperatively calculates on protected data.

In SC protocol, $RG$ generates the shares of parameters $\ell, r, r'$ by uploading them as shown in Figure 4(b). After that, we calculate the subtraction of $\langle x\rangle$ and $\langle y\rangle$, where the MSS subtraction $\langle \cdot - \cdot\rangle$ can regard as adding the inverse element of the subtrahend to the minuend through MSSAdd. Since the additive inverse always exists in modulus, the negative can represent by shares in MSS under modulo $p$. And finally, we imply the difference into $\langle e\rangle$ and $\langle h\rangle$ through consistent changes, which the reconstruction could reveal the relationship but not the exact gap between the secrets.

**Protocol 4** Secure Comparison (SC)

$CS_1, CS_2, P_i$ **Input**: $\langle x\rangle = ([\alpha], [\alpha x])$, $\langle y\rangle = ([\beta], [\beta y])$
$CS_1, CS_2, P_i$ **Output**: 1 if $(x > y)$ or 0 if $(x \leq y)$

1. $RG$ produces $\langle \ell\rangle$, where $x, y \leq \ell$ ($\ell$ is a public parameter).
2. $CS_1, CS_2, P_i$ collaboratively compute $\langle z\rangle = \langle x - y + \ell\rangle$.
3. $RG$ uniformly picks two positive integers $r$ and $r'$, where $2\ell r + r' < q$, and then generates $\langle r\rangle, \langle r'\rangle$.
4. $CS_1, CS_2, P_i$ collaboratively compute $\langle e\rangle = \langle z * r + r'\rangle$ and $\langle h\rangle = \langle \ell * r + r'\rangle$.
5. $CS_1, CS_2, P_i$ collaboratively reconstruct $e$ and $h$. Then, outputs 1 if $e > h$ or 0 if not.

## VI. Application: Privacy-preserving $k$NN based on multi-secret sharing

In this section, we demonstrate our presented building blocks to realize an application, privacy-preserving $k$NN. In fact, these building blocks can be expected for different purposes of functionality. However, privacy-preserving $k$NN is selected as a conceptually simple illustration since they can directly map the requirements of running $k$NN. We postpone showing some discussions of the other applications by applying standard techniques to Section VIII-D.

During the process of privacy-preserving $k$NN, we outsource the main calculation work to the cloud servers and compute the dataset and the query data in the shared state to keep data information from leakage. Additionally, each participant only holds a single share that provides access to the data, which is universal for the entire calculation and maintains the cloud assistance multi-secret sharing throughout the service. Our privacy-preserving k-nearest neighbor classification consists of four stages, i.e., system initialization, data outsourcing, request generation, and service processing.

### *A. System Initialization*

This system contains four types of entities, namely the dealer, dual-cloud servers $(CS_1, CS_2)$, multiple participants $(P_1, P_2, ..., P_n)$, and a randomness generator $(RG)$. First of all, the dealer provides the dataset $D = (d, l)$ for the classification service, where $d = \{d_{1,1}, ..., d_{v,m}\}$ is the total data from the $v$ instances and $m$ attributes, and $l = \{l_1, l_2, ..., l_v\}$ is the label for each instance. Then, the participant may have some queries $Q = \{Q_1, Q_2, ..., Q_u\}$, where $u$ is the number of the request and $q = \{q_1, q_2, ..., q_m\}$ is the query data for each request. Since we assume that the data is composed of rounding the floating-point numbers into integers, the modulo calculation always holds during secret sharing. Moreover, the dealer adds 1 into the dataset as an assistant value for MSS calculation. After that, the dealer generates the participant shares $S = \{S_1, S_2, ..., S_n\}$ and sends $S_i$ to participant $P_i$, respectively.

### *B. Data Outsourcing*

With the participant shares $S$, the dealer can convert the dataset into public shares in MSS and outsources the $k$NN classification service on cloud servers. To protect dataset $D$, the dealer randomly picks the masks and multiplies them with the data in the dataset to produce the masked data. Then, the dealer generates public shares of the protected data by multi-secret sharing and distributes them to $CS_1$ and $CS_2$.

Carrying out function $Share_{(S)}(x, t) = ([\alpha]_{pub}, [\alpha x]_{pub})$ with dataset $D = (d, l)$. We denote the MSS shares related to secret $x$ as $\langle x\rangle$ that consists of $[\alpha]_{pub}$, $[\alpha x]_{pub}$ and $S_i (= [\alpha]_i = [\alpha x]_i)$, which are shares in $CS_1$, $CS_2$ and participant $P_i$, respectively. Therefore, the system has $\langle \boldsymbol{d}\rangle = \{\langle d_{1,1}\rangle, ..., \langle d_{v,m}\rangle\}$ and $\langle \boldsymbol{l}\rangle = \{\langle l_1\rangle, \langle l_2\rangle, ..., \langle l_v\rangle\}$ after outsourcing. From now on, the dealer can get offline in our system.

### *C. Request Generation*

After outsourcing the model to cloud servers, participants can upload query data and launch the request with participants that meet the threshold $t$ for completing calculations in the classification service. Although multi-secret sharing asks the share generation to involve participant shares $S$, we can form the MSS shares of the query based on some existing MSS shares.

To securely upload the query $q$, we add an assistance value 1 at the system initialization that becomes $\langle 1\rangle$ in our system. Then, we rely on $RG$ to transform $\langle 1\rangle$ into $\langle \boldsymbol{q}\rangle = \{\langle q_1\rangle, ..., \langle q_m\rangle\}$ through the SD protocol. Since we can upload the data without knowing other shares, we can generate the request anytime and perform calculations with the protected dataset in the system.

### D. Service Processing

Finally, we have the dataset $\langle \boldsymbol{d} \rangle, \langle \boldsymbol{l} \rangle$ and query data $\langle \boldsymbol{q} \rangle$ in our system. Next, we can accomplish the privacy-preserving $k$NN classification service by applying our proposed protocols. In Algorithm 1, we use the SC protocol to compare the relationship between the attributes of the query data and each dataset instance. Then, we can get the positive differences $\langle \triangle \rangle$ without learning any information about the query data and dataset. After that, we aggregate the differences and define the distance between the query and each instance label.

Once we have the distance, we can search the nearest neighbors through secure comparison, where distances are also an MSS share that sums the positive differences $\langle \triangle \rangle_i$ through MSSAdd. After finding the $k$-th nearest neighbors, we reconstruct the selected labels, where $CS_1$ obtains the mask and $CS_2$ gets the masked data. Then, we can determine which category belongs to the query. Before we end the classification, we can clear the calculation process by OpRec.$delete()$ and RandRec.$delete()$. Thus, we can cast off the accumulation of residual data that we no longer need for persistent computation.

**Algorithm 1:** Privacy-preserving $k$NN Classification

```
Input: dataset $\langle \boldsymbol{d} \rangle = \{\langle d_{1,1} \rangle, ..., \langle d_{v,m} \rangle\}$,
$\langle \boldsymbol{l} \rangle = \{\langle l_1 \rangle, \langle l_2 \rangle, ..., \langle l_v \rangle\}$ and query
$\langle \boldsymbol{q} \rangle = \{\langle q_1 \rangle, \langle q_2 \rangle, ..., \langle q_m \rangle\}$.
Output: respond category $c$.
for $i = 1$ to $v$ do
    for $j = 1$ to $m$ do
        if SC($\langle d_{i,j} \rangle, \langle q_j \rangle$) then
            $\langle \triangle_i \rangle \leftarrow \langle \triangle_i + d_{i,j} - q_j \rangle$;
        else
            $\langle \triangle_i \rangle \leftarrow \langle \triangle_i + q_j - d_{i,j} \rangle$;
        end
    end
    $dist[i] \leftarrow \langle \triangle_i \rangle$;
    $label[i] \leftarrow \langle l_i \rangle$;
end
for $i = 1$; $i < v$ ; $i++$ do
    $min \leftarrow i$;
    for $j = i + 1$ to $v$ do
        if SC($dist[min], dist[j]$) then
            $min \leftarrow j$;
        end
    end
    Swap $dist[i]$ and $dist[min]$;
    Swap $label[i]$ and $label[min]$;
    if $i == k$ then
        Reconstruct $label[1 : k]$;
        Select the majority label as category $c$;
        break;
    end
end
Runs OpRec.$delete()$;
Runs RandRec.$delete()$;
return $c$;
```

## VII. Security Analysis

In this section, we analyze the security of the proposed protocols, which serve as MSS tools for outsourcing services. To prove the composition is secure, we adopt the framework of universal composability (UC) [29] and show the view of the semi-honest adversaries. According to the threat model, we may consider the adversary with a view involving their inputs, the received messages, and the information from the colluding parties. Although the adversary may learn some protected data through the protocols, secrets never leak during persistent computations.

### A. Security of Composition Protocol

**Functionality 1** $\mathcal{F}_{SD}$: $\mathcal{F}_{SD}$ interacts with $RG, CS, P_i$ and an adversary $\mathcal{S}$.

1. Upon receiving a message $([y]_i, [a]_i, [b]_i, [c]_i)$ from $RG$, send the message $([y]_i, [a]_i, [b]_i, [c]_i)$ to $P_i, CS$ and $\mathcal{S}$.
2. Upon receiving a message $([x-a]_i, [y-b]_i)$ from $P_i$, send the message $([x-a]_i, [y-b]_i)$ to $CS$ and $\mathcal{S}$.
3. Upon receiving a message $(\epsilon, \rho)$ from $CS$, send the message $(\epsilon, \rho)$ to $P_i$ and $\mathcal{S}$.

**Theorem 1** (Security of $\Pi_{SD}$). *$\Pi_{SD}$ UC-realizes $\mathcal{F}_{SD}$.*

Proof. To explain the privacy of $\Pi_{SD}$, we first consider the view of $RG$. While $\mathsf{View}_{RG} = (y, a, b, c)$, $RG$ has no information about $xy$ and cannot learn $x$ in $\Pi_{SD}$. Then, we consider an adversary $\mathcal{A}$ colludes with less than $t$ participants, where $\mathsf{View}_{\mathcal{A}} = ([x]_i, [y]_i, [a]_i, [b]_i, [c]_i, \epsilon, \rho, [xy]_i)$ and $i = \{j_1, ..., j_{t-1}\}$. Since secret sharing is information-theoretic security, $\mathcal{A}$ does not have enough share to recover the real values of $a, b$, and $xy$. Consequently, $x$ and $y$ are secure even if $\epsilon$ and $\rho$ is a known in $\Pi_{SD}$. Altogether, $\mathcal{A}$ causes nothing leakage and is a perfect simulation for $\mathcal{S}$ in an ideal world. Therefore, we can state $\Pi_{SD}$ UC-realizes $\mathcal{F}_{SD}$.

**Functionality 2** $\mathcal{F}_{MSSMul}$: $\mathcal{F}_{MSSMul}$ interacts with $CS_1, CS_2, P_i$ and $\mathcal{S}$.

1. Run $\mathcal{F}_{SD}$ to refresh $\langle x \rangle$ = $([\alpha], [\alpha x])$ and $\langle y \rangle$ = $([\beta], [\beta y])$.
2. Upon receiving a message $([r_1\alpha]_i)$ from $P_i$, send $([r_1\alpha]_i)$ to $CS_1$ and $\mathcal{S}$.
3. Upon receiving a message $([r_2\beta y]_i)$ from $P_i$, send $([r_2\beta y]_i)$ to $CS_2$ and $\mathcal{S}$.
4. Upon receiving a message $(r_1\alpha, r_2\beta y)$ from $CS_1$ and $CS_2$, send the message $(r_1\alpha, r_2\beta y)$ to $CS_1, CS_2, P_i$ and $\mathcal{S}$.

**Theorem 2** (Security of $\Pi_{MSSMul}$). *$\Pi_{MSSMul}$ UC-realizes $\mathcal{F}_{MSSMul}$ in the ($\mathcal{F}_{SD}$)-hybrid model.*

Proof. We define $\Pi_{MSSMul}$ in the ($\mathcal{F}_{SD}$)-hybrid model. Since $\Pi_{SD}$ randomly selects $a$ and $b$, the environment machine $\mathcal{Z}$ cannot distinguish the worlds from the timely information. Thus, $\Pi_{MSSMul}$ can UC-emulate $\Pi_{SD}$ in the ($\mathcal{F}_{SD}$)-hybrid model. Moreover, we observe the views of $CS_1$ and $CS_2$, respectively. For $\mathsf{View}_{CS_1} = ([\alpha]_{pub}, [\beta]_{pub},$

$r_1\alpha, [r_2\beta]_{pub}, r_2\beta y\ \ , [r_1 r_2\alpha\beta]_{pub})$, $CS_1$ cannot obtain $y$ since participants do not offer $[r_2\beta]_i$. For $\mathsf{View}_{CS_2} = ([\alpha x]_{pub}, [\beta y]_{pub}, r_1\alpha, [r_1\alpha x]_{pub}, r_2\beta y, [r_1 r_2\alpha\beta xy]_{pub})$, $CS_2$ cannot obtain $x$ since participants do not offer $[r_1\alpha x]_i$. Hence, $x$ and $y$ keep confidential while revealing $r_1\alpha$ and $r_2\beta y$. Not to mention that $r_1 r_2\alpha\beta$ and $r_1 r_2\alpha\beta xy$ can only recover on the participant side with enough shares. In summary, adversary $\mathcal{A}$ is impossible to obtain additional information, which well-simulates $\mathcal{S}$ in the ideal world. Therefore, we can prove $\Pi_{MSSMul}$ UC-realizes $\mathcal{F}_{MSSMul}$ with the universal composition theorem.

**Functionality 3** $\mathcal{F}_{MSSAdd}$: $\mathcal{F}_{MSSAdd}$ interacts with $CS_1, CS_2, P_i$ and $\mathcal{S}$.

1. Run $\mathcal{F}_{SD}$ to refresh $\langle x\rangle = ([\alpha], [\alpha x])$ and $\langle y\rangle = ([\beta], [\beta y])$.
2. Upon receiving a message $([r_1\alpha]_i)$ from $P_i$, send $([r_1\alpha]_i)$ to $CS_1$ and $\mathcal{S}$.
3. Upon receiving a message $([r_2\beta]_i)$ from $P_i$, send $([r_2\beta]_i)$ to $CS_1$ and $\mathcal{S}$.
4. Upon receiving a message $(r_1\alpha, r_2\beta)$ from $CS_1$, send the message $(r_1\alpha, r_2\beta)$ to $CS_1, CS_2, P_i$ and $\mathcal{S}$.

**Theorem 3** (Security of $\Pi_{MSSAdd}$)**.** *$\Pi_{MSSAdd}$ UC-realizes $\mathcal{F}_{MSSAdd}$ in the ($\mathcal{F}_{SD}$)-hybrid model.*

Proof. Similar to Theorem 2, we define $\Pi_{MSSAdd}$ in the ($\mathcal{F}_{SD}$)-hybrid model. Since $\Pi_{SD}$ randomly selects $a$ and $b$, the real word and the ideal world are indistinguishable for $\mathcal{Z}$. Then, $\Pi_{MSSAdd}$ can UC-emulate $\Pi_{SD}$ in the ($\mathcal{F}_{SD}$)-hybrid model. After that, we observe the views of $CS_1$ and $CS_2$, respectively. $CS_1$ obtains the refreshed masks, where $\mathsf{View}_{CS_1} = ([\alpha]_{pub}, [\beta]_{pub}, r_1\alpha, r_2\beta, [r_1 r_2\alpha\beta]_{pub})$. $CS_2$ will learn nothing because the refreshed masked data remain as shares, where $\mathsf{View}_{CS_2} = ([\alpha x]_{pub}, [\beta y]_{pub}, [r_1\alpha x]_{pub}, [r_2\beta y]_{pub}, r_1\alpha, r_2\beta, [r_1 r_2\alpha\beta(x+y)]_{pub})$. Since a part of the secret stays secure, performing calculations on shares will not disclose the real value in MSS. To conclude, adversary $\mathcal{A}$ learns nothing about the secret as long as the participant does not reach the threshold. As $\mathcal{A}$ perfectly simulates $\mathcal{S}$ in the ideal world. Therefore, we can show $\Pi_{MSSAdd}$ UC-realizes $\mathcal{F}_{MSSAdd}$ in the universal composition theorem.

**Functionality 4** $\mathcal{F}_{SC}$: $\mathcal{F}_{SC}$ interacts with $CS_1, CS_2, P_i$ and an adversary $\mathcal{S}$.

1. Run $\mathcal{F}_{SD}$ to produce $\langle \ell\rangle$.
2. Run $\mathcal{F}_{MSSAdd}$ to compute $\langle z\rangle$.
3. Run $\mathcal{F}_{SD}$ to generate $\langle r\rangle$ and $\langle r'\rangle$.
4. Run $\mathcal{F}_{MSSMul}$ and $\mathcal{F}_{MSSAdd}$ to computes $\langle e\rangle = \langle z * r + r'\rangle$ and $\langle h\rangle = \langle \ell * r + r'\rangle$.
5. Upon receiving a message $(\langle e\rangle, \langle h\rangle)$ from $CS_1$ and $CS_2$, send the message $(\langle e\rangle, \langle h\rangle)$ to $P_i$ and $\mathcal{S}$.
6. Upon receiving a message $(output)$ from $P_i$, send the message $(output)$ to $CS_1, CS_2, P_i$ and $\mathcal{S}$.

**Theorem 4** (Security of $\Pi_{SC}$)**.** *$\Pi_{SC}$ UC-realizes $\mathcal{F}_{SC}$ to complete secure comparison.*

Proof. $\Pi_{SC}$ invokes secure protocols as subroutines, which we already prove UC-secure in Theorem 1, Theorem 2 and Theorem 3. Looking at the proving process above, we could find $\Pi_{MSSMul}$ and $\Pi_{MSSAdd}$ pick random numbers $r_1$ and $r_2$ for refreshing secrets through $\Pi_{SD}$. Since $\Pi_{SD}$ also constructs from randomness parameters, $\Pi_{SC}$ can UC-emulate these subroutines. Next, the view of adversary $\mathcal{A}$ is $(\langle x\rangle, \langle y\rangle, \langle \ell\rangle, \langle z\rangle, \langle r\rangle, \langle r'\rangle, e, h, output)$. Since multi-secret sharing carries on the information-theoretic security from secret sharing, $\mathcal{A}$ could reconstruct nothing in the view, and knowing $e$ and $h$ is helpless for getting more information. Because $\mathcal{Z}$ cannot obtain any clues to distinguish the real-world execution from the ideal-world execution, $\Pi_{SC}$ securely realizes functionality $\mathcal{F}_{SC}$ for secure comparison. To sum up, we can demonstrate $\Pi_{SC}$ UC-realizes $\mathcal{F}_{SC}$ when the subroutines interact as in the real-life model.

### B. Security of kNN Classification

We have analyzed the composition security of the proposed protocols and guarantee that privacy will not leak during the persistent computations. Eventually, we take privacy-preserving $k$NN as an example and briefly describe the security of an MSS outsourcing service.

First, the phases of system initialization and data outsourcing are non-interactive, so we know that the system is safe through the distribution settings. Then, we upload the query data with SD protocol in the request generation phase. As the $\Pi_{SD}$ is proved UC-secure, so is the data uploading. Finally, the classification consists of several persistent computations and numerous secure comparisons, where the protocols proved UC-secure in the above analysis. Afterward, we can consider an adversary $\mathcal{A}$ who colludes with participants less than $t$ to get access to dataset $D$ and query data $x$. Let $\pi$ be the classification process, and function $f$ represents the privacy leakages. We can get the probabilities below.

- $Pr[\mathcal{A}^{\pi}(D) \rightarrow f(x)] < \mathsf{negl}$
- $Pr[\mathcal{A}^{\pi}(x) \rightarrow f(D)] < \mathsf{negl}$

While the adversary colludes with someone who knows the dataset, they cannot learn the query data through the classification service. On the other hand, classifying the query data launched from the adversary will not cause privacy leakage of the outsourced dataset. Therefore, we can claim that the ciphertext does not reveal any information about the plaintext through the outsourcing service based on MSS protocols.

## VIII. Evaluation

To evaluate our designed protocols, we theoretically analyze the cost of each function and measure the performance by massive persistent computations from $k$NN classification in the experiment.

### A. Theoretical Analysis

We use $(n, t)$ to denote the $t$-out-of-$n$ multi-secret sharing, where $n$ is the number of participants and $t$ is the threshold $(t \leq n)$ for each secret. In our scheme, $CS_1$, $CS_2$, and $RG$

Table III: Complexities (computation, communication, storage) of each protocol.

$n$: participants, $t$: thresholds

| Protocol | $P$ | $CS_1$ | $CS_2$ | $RG$ |
|---|---|---|---|---|
| SD | $O(1)$, $O(1)$, N/A | $O(n-t)$, $O(1)$, N/A | $O(n-t)$, $O(1)$, N/A | $O(1)$, $O(n)$, $O(n)$ |
| MSSMul | $O(1)$, $O(1)$, N/A | $O(n-t)$, $O(n)$, $O(1)$ | $O(n-t)$, $O(n)$, N/A | $O(1)$, $O(n)$, $O(n)$ |
| MSSAdd | $O(1)$, $O(1)$, N/A | $O(n-t)$, $O(n)$, $O(1)$ | N/A, N/A, N/A | $O(1)$, $O(n)$, $O(n)$ |
| SC | $O(1)$, $O(1)$, N/A | $O(n-t)$, $O(n)$, $O(1)$ | $O(n-t)$, $O(n)$, N/A | $O(1)$, $O(n)$, $O(n)$ |

have the communication complexity of $O(n)$ through sending and receiving messages among $n$ participants. Since there is no communication between participants, the communication cost of the participant is $O(1)$. For secure computations, each entity implies the operation by locally calculating their shares, so the participant has the computation cost of $O(1)$, and the cloud servers that modify $n-t+1$ shares of a secret have the computational complexity of $O(n-t)$.

Assuming that we perform $k$NN classification of a single query to an outsourced dataset with $v$ instances and $m$ attributes, the storage complexity of each cloud server is $O(v \cdot m \cdot (n-t))$ and the storage overhead of the participant is always 1 in our system. Since we upload the query through SD protocol, RandRec adds about three times $2n-t+1$ shares for each Beaver triple. Hence, the storage complexity of $RG$ is $O(m \cdot n)$ in a request generation with $m$ query data.

Table III is the complexity of construction for each protocol. While SC consists of three MSS share generations and several MSS operations, the complexities of SC are equal to the highest order of adding up the operations together. Accordingly, we can evaluate the classification by breaking the algorithm into two parts. The first part takes $O(v \cdot m)$ times to calculate the distance between the query and the dataset, mainly composed of SC, MSSMul, and MSSAdd. The second part uses $O(k \cdot v)$ times to compare the distance shares and find the $k$ nearest labels through the SC protocol. At last, we can clean up the calculations that will no longer need after this classification, and the system can restore to the initial state.

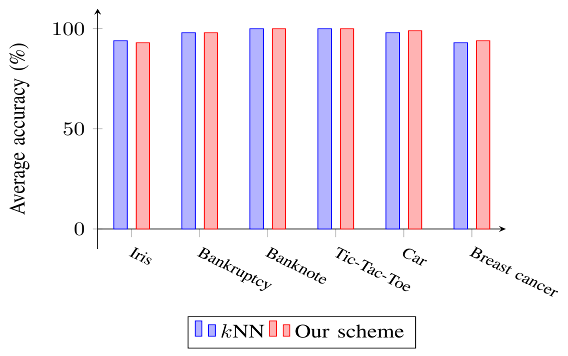


Figure 7: Average accuracy in different datasets.

## B. Experimental Results

We set up the experiments on a desktop computer with Windows 11 Pro operating system, Intel(R) Core(TM) i7-8700 3.20GHz CPU, and 16GB memory as specification. Then, we realize our scheme with python 3.9.2 and simulate all actions of the privacy-preserving $k$NN classification service on several datasets. We point out our work applies to various tasks by choosing datasets with different backgrounds from the UCI Machine Learning Repository. More specifically, we take iris [3] to test pattern recognition, bankruptcy [4] to build an expert decision, and banknote [5] to authenticate from extracted features of images. Besides, we show the possibility of our scheme in some industries through tic-tac-toe [6], car [7], and breast cancer [8]. These datasets are preprocessed through ordinal encoding and rounding numbers to integers in our experiments.

In the implementation, we pick a large prime greater than any secret of the dataset to fulfill the MSS in our system. Then, we randomly sample 10 test cases for requests and evaluate the average with epochs of 10. Figure 7 shows the accuracy comparison between the ordinary $k$NN and our outsourced $k$NN. Thus, we can prove the applicability by highlighting the close accuracy of the outsourced service based on our scheme. Figure 8 and Figure 9 demonstrate the performance under different settings. Looking up to Table IV, we can observe the execution time mainly grows with the number of participants and the total data of the dataset. Although the execution time increases with the size of the participants, performance remains the same on the varying threshold. Therefore, there is no trade-off between security and efficiency as we raise the threshold to form a more secure system that requires higher access rights.

Table IV: Parameters of each dataset.

| Dataset | Instances | Attributes | Classes | Total Data |
|---|---|---|---|---|
| Iris | 150 | 4 | 3 | 600 |
| Bankruptcy | 250 | 6 | 2 | 1500 |
| Banknote | 1372 | 4 | 2 | 5488 |
| Tic-Tac-Toe | 958 | 9 | 2 | 8622 |
| Car | 1728 | 6 | 4 | 10368 |
| Breast cancer | 569 | 30 | 2 | 17070 |

[3]Iris. https://doi.org/10.24432/C56C76
[4]Qualitative_Bankruptcy. https://doi.org/10.24432/C52889
[5]Banknote Authentication. https://doi.org/10.24432/C55P57
[6]Tic-Tac-Toe Endgame. https://doi.org/10.24432/C5688J
[7]Car Evaluation. https://doi.org/10.24432/C5JP48
[8]Breast Cancer Wisconsin (Diagnostic). https://doi.org/10.24432/C5DW2B

Table V: Cost comparison with fully homomorphic encryption (CKKS), secret sharing, and our MSS on Iris dataset.

| Scheme | Client | | Server | |
|---|---|---|---|---|
| | Ciphertext Size | Execution Time | Ciphertext Size | Execution Time |
| CKKS [15] | 1.65 MB | High | 247.5 MB | High |
| SS [22] | 1.46 KB | middle | 1.46 KB | Low |
| MSS (Ours) | 10 byte | Low | 1.46 KB | middle |

### C. Comparisons with Related works

Eventually, we show the strengths of our scheme over other previous solutions by evaluating them on the Iris dataset. Table V briefly summarizes the cost comparison in the three schemes. Since the query we consider at the client is an instance consisting of 4 attributes and the dataset is 150 instances in the server. The ciphertext size transferred from the client to the server in CKKS is 1.65 MB. Meanwhile, the data size transferred between the client and the server in SS is 1.46 KB, and in our approach is 10 bytes. In terms of the client, our approach significantly outperforms others. Note that to provide a fair comparison, we use parameters with a similar configuration (the prime number is set as $2^{19}-1$ and assuming the participants and threshold as 2) for the evaluation.

### D. Potential Applications

Essentially, we aim to adopt MSS to work within practical constraints scenarios. Thus, we pay attention to operations inside the crypto tools rather than their applications in this work. While privacy-preserving $k$NN may be perceived as less exciting tasks, they are crucial to ensure applications runnable in practical data science. Our protocol focuses on secure computation for resource-restricted devices, which means that it is independent of any specific tasks and possible to combine with any arithmetic technique. Here, we give a trivial solution to transform diverse jobs into a realization of arithmetic operations. By the Lagrange interpolation, we can easily convert the input and output into polynomial form. After that, we can upload the coefficients and complete the secure computation on the DCMSS. Besides, some inspiring ideas on polynomial approximation can referred to [16]. Finally, several frequently discussed topics, such as AES Sbox, Maxpool, and ReLU, are highly anticipated to be implemented through this general approach. We have a partial demonstration involving AES Sbox [9] available on GitHub. The DCMSS scheme can enhance the usage of cryptography and related applications, yet we will conduct further research for a detailed analysis of this improvement in the future.

## IX. Conclusions

In this paper, we proposed a persistent computing architecture based on multi-secret sharing and constructed some privacy-preserving operation protocols above it. Our scheme can achieve cloud assistance multi-secret sharing (CAMSS) by exchanging the storage costs with communication and computation, which helps to form a secure and efficient computing pattern on resource-restricted devices. We ensure computational feasibility and security for this form and build a $k$NN classification to evaluate the performance. In the future, we would like to extend this research to more applications, discuss and address particular security issues, and create other possible operational protocols.

[9]Demo (AES Sbox): https://github.com/cislab-ntut/AES_sbox2poly

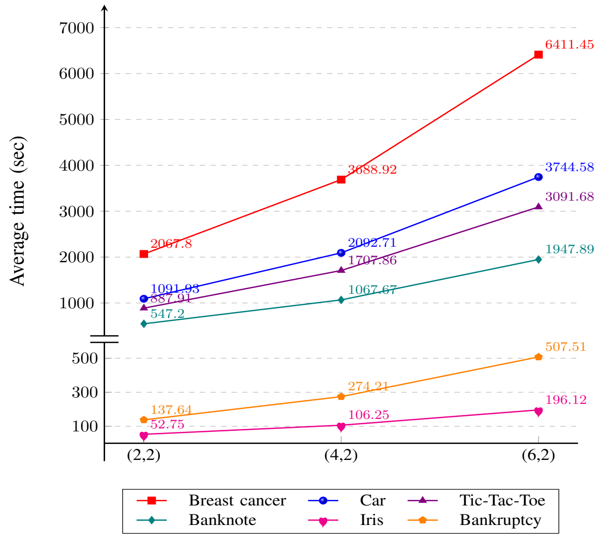


Figure 8: Average execution time with different numbers of participants.

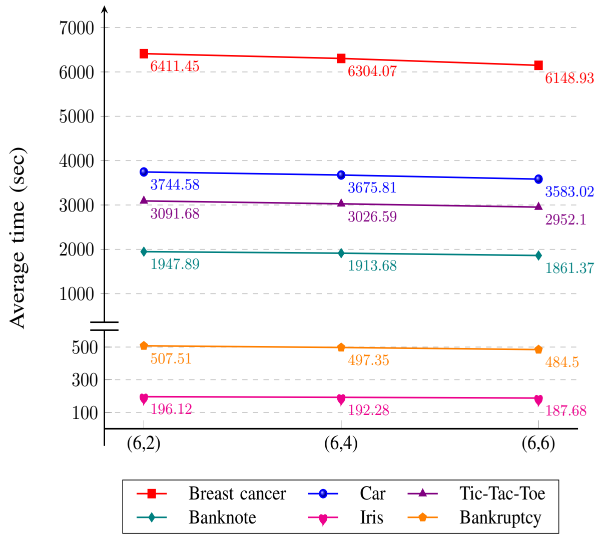


Figure 9: Average execution time with varying thresholds.

## Acknowledgment

We highly appreciate the insightful and helpful comments of the reviewers on the presentation of this paper. This work was supported by National Science and Technology Council of Taiwan (Nos. 112-2221-E-027-069 and 112-2634-F-027-001-MBK).

## References

[1] D. Richins, D. Doshi, M. Blackmore, A. Thulaseedharan Nair, N. Pathapati, A. Patel, B. Daguman, D. Dobrijalowski, R. Illikkal, K. Long, D. Zimmerman, and V. Janapa Reddi, "Missing the forest for the trees: End-to-end ai application performance in edge data centers," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 515–528.

[2] M. Hind, D. Wei, M. Campbell, N. C. F. Codella, A. Dhurandhar, A. Mojsilović, K. Natesan Ramamurthy, and K. R. Varshney, "Ted: Teaching ai to explain its decisions," in *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, ser. AIES '19, 2019, p. 123–129.

[3] H. W. Lim, G. S. Poh, J. Xu, and V. Chittawar, "PrivateLink : Privacy-preserving integration and sharing of datasets," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 564–577, 2019.

[4] Q. Xue, Y. Zhu, J. Wang, and X. Li, "Distributed set intersection and union with local differential privacy," in *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*, 2017, pp. 198–205.

[5] V. M. Shelake and N. M. Shekokar, "Smspprl: A similarity matching strategy for privacy preserving record linkage," in *2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, 2020, pp. 481–485.

[6] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[7] M. Franklin and M. Yung, "Communication complexity of secure computation (extended abstract)," in *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '92. New York, NY, USA: Association for Computing Machinery, 1992, p. 699–710. [Online]. Available: https://doi.org/10.1145/129712.129780

[8] D. Chen, W. Lu, W. Xing, and N. Wang, "An efficient verifiable threshold multi-secret sharing scheme with different stages," *IEEE Access*, vol. 7, pp. 107 104–107 110, 2019.

[9] A. Wigderson, M. Or, and S. Goldwasser, "Completeness theorems for noncryptographic fault-tolerant distributed computations," in *Proceedings of the 20th Annual Symposium on the Theory of Computing (STOC'88)*, 1988, pp. 1–10.

[10] T. Shingu., K. Iwaumura., and K. Kaneda., "Secrecy computation without changing polynomial degree in shamir's (k, n) secret sharing scheme," in *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications - DCNET, (ICETE 2016)*, 2016, pp. 89–94.

[11] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009, crypto.stanford.edu/craig.

[12] R. Koseki, A. Ito, R. Ueno, M. Tibouchi, and N. Homma, "Homomorphic encryption for stochastic computing," *Journal of Cryptographic Engineering*, vol. 13, no. 2, pp. 251–263, Jun 2023. [Online]. Available: https://doi.org/10.1007/s13389-022-00299-6

[13] B. R. Gaines, "Stochastic computing systems," *Advances in Information Systems Science: Volume 2*, pp. 37–172, 1969.

[14] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology – ASIACRYPT 2017*, T. Takagi and T. Peyrin, Eds. Cham: Springer International Publishing, 2017, pp. 409–437.

[15] Z. Jiang, W. Wang, and Y. Liu, "FLASHE: additively symmetric homomorphic encryption for cross-silo federated learning," *CoRR*, vol. abs/2109.00675, 2021. [Online]. Available: https://arxiv.org/abs/2109.00675

[16] E. Hesamifard, H. Takabi, M. Ghasemi, and R. N. Wright, "Privacy-preserving machine learning as a service." *Proc. Priv. Enhancing Technol.*, vol. 2018, no. 3, pp. 123–142, 2018.

[17] H. Kaur and P. Khanna, "Privacy preserving remote multi-server biometric authentication using cancelable biometrics and secret sharing," *Future Generation Computer Systems*, vol. 102, pp. 30–41, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X18330553

[18] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, "Verifiable secret sharing and achieving simultaneity in the presence of faults," in *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*. IEEE, 1985, pp. 383–395.

[19] M. Stadler, "Publicly verifiable secret sharing," in *Advances in Cryptology — EUROCRYPT '96*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 190–199.

[20] Z. Ren, X. Cheng, M. Fan, J. Zhang, and C. Hong, "Communication efficient secret sharing with dynamic communication-computation conversion," in *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*, 2023, pp. 1–10.

[21] A. Shivhare, M. K. Maurya, J. Sarif, and M. Kumar, "A secret sharing-based scheme for secure and energy efficient data transfer in sensor-based iot," *The Journal of Supercomputing*, vol. 78, no. 15, pp. 17 132–17 149, Oct 2022. [Online]. Available: https://doi.org/10.1007/s11227-022-04533-0

[22] J.-K. Yang, K.-C. Huang, C.-Y. Chung, Y.-C. Chen, and T.-W. Wu, "Efficient privacy preserving nearest neighboring classification from tree structures and secret sharing," in *ICC 2022 - IEEE International Conference on Communications*, 2022, pp. 5615–5620.

[23] V. Attasena, J. Darmont, and N. Harbi, "Secret sharing for cloud data security: a survey," *The VLDB Journal*, vol. 26, no. 5, pp. 657–681, Oct 2017. [Online]. Available: https://doi.org/10.1007/s00778-017-0470-9

[24] A. Nag, S. Choudhary, S. Dawn, and S. Basu, "Secure data outsourcing in the cloud using multi-secret sharing scheme (msss)," in *Proceedings of the First International Conference on Intelligent Computing and Communication*, J. K. Mandal, S. C. Satapathy, M. K. Sanyal, and V. Bhateja, Eds. Singapore: Springer Singapore, 2017, pp. 337–343.

[25] Z. Tang, "Secret sharing-based iot text data outsourcing: A secure and efficient scheme," *IEEE Access*, vol. 9, pp. 76 908–76 920, 2021.

[26] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Advances in Cryptology — CRYPTO '91*, J. Feigenbaum, Ed., 1992, pp. 420–432.

[27] N. Bhatia and Vandana, "Survey of nearest neighbor techniques," *CoRR*, vol. abs/1007.0085, 2010. [Online]. Available: http://arxiv.org/abs/1007.0085

[28] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 19–38.

[29] R. Canetti, "Universally composable security: a new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, 2001, pp. 136–145.