Private Weighted Sum Aggregation

Andreea B. Alexandru, *Member, IEEE*, and George J. Pappas, *Fellow, IEEE*

***Abstract*—As large amounts of data are circulated both from users to a cloud server and between users, privately aggregating the shared data is critical. This article considers the problem of *private weighted sum aggregation with secret weights*, where an aggregator wants to compute the weighted sum of the local data of some agents. Based on the privacy requirements posed on the weights, there are different secure multiparty computation schemes exploiting the knowledge structure. First, we review schemes for when each agent has a local private value and local private weight, and when agents have a local private value, but the aggregator holds the corresponding weights. Our main focus is the more general case, where the weights are known neither by the agents nor by the aggregator—they are generated and kept private by a system operator, and the aggregator has to compute the weighted sum without learning the agents' data or the weights. We give solutions that achieve aggregator obliviousness and design more efficient communication and computation strategies for multidimensional data by batching the data into fewer ciphertexts. Finally, we implement our schemes and discuss the numerical results.**

***Index Terms*—Cryptography, data privacy, distributed systems, security.**

I. Introduction

THE recent technological advances in communication speed and deployment of millions of devices have fostered the adoption of distributed computing frameworks. In turn, such frameworks require aggregating services in order to utilize the data collected for specific causes. Even as a step in distributed algorithms, aggregation shifts the gears from a decentralized nature that inherently guarantees more privacy, to a centralized approach that poses severe privacy challenges.

Of particular interest is the general problem of weighted sum aggregation, in which an aggregating party needs to collect and sum contributions from a number of agents–the contributions consist of local data weighted by some relevant quantities. There is a wealth of examples spanning various research areas that require the computation of weighted aggregates: (a) Decentralized and cooperative linear control for multiagent systems [1]–[3]; (b) Graph neural networks [4], [5] and collaborative inference [6], [7]; (c) Average consensus [8], [9]; (d) Federated learning and aggregation of linear inference results [10], [11]; (e) Energy price aggregation and management [12], [13], vehicle tolls collection [14], [15]. Each of the above examples can pose different privacy requirements on the local data of the agents, as well as on the corresponding weights. For example, average consensus (c) and federated learning (d) can be seen as a *private sum aggregation (pSA)* problem—weights are public or known by the agents, hence, the agents can absorb them in their private data. In some price collection instances (e), the prices can depend on private information known at the aggregator and can vary dynamically, so the aggregator knows the price weights, while the agents do not, which can be modeled as a *private weighted sum aggregation with centralized weights* at the aggregator. There are cases where a system operator has invested resources into computing the control gains for a distributed system and wants to keep them private from both the agents and the aggregator, who needs to compute a linear control without knowing neither the local agent's states nor the gains. Similar privacy requirements are in place for secure inference, where a service provider has trained a proprietary model on its own data and wants to keep it private while allowing it to be deployed. In these latter two cases (a), (b), not revealing the proprietary weights to the participants prevents unintended privacy loss when an agent is captured in an adversarial environment and its memory seized or some of the agents are rival companies. This competitive–collaborative scenario, where there is no loss of private information to adversaries, yet the computation is allowed to proceed, is the focus of the current article, and can be formulated as a *private weighted sum aggregation with hidden weights* problem.

In the context of (a), linear distributed control with homomorphically encrypted gains was addressed by [16]–[18], with [18] touching also on (b). We will elaborate and improve on these works in Section V. Concerning (c), there is a body of research that targets the privacy of the local initial states of the agents achieving consensus, using partially homomorphic encryption or differential privacy: see [19]–[23] and the references within. For (d) and (e), works such as [24]–[28] provide private solutions for pSA, touching on a large base of cryptographic tools, such as secret sharing, threshold homomorphic encryption, and differential privacy.

As illustrated above, under particular knowledge distribution and privacy demands on the weights and agents' local data, a wide range of problems can be written in a unified formulation as private weighted sum aggregation problems. Our main contribution is to offer a private solution for the general case

Manuscript received May 5, 2021; revised June 11, 2021; accepted June 16, 2021. Date of publication July 7, 2021; date of current version May 26, 2022. Recommended by Associate Editor P. Cheng. *(Corresponding author: Andreea B. Alexandru.)*

Andreea B. Alexandru is with the Department of Computer Science, University of Maryland, College Park, MD 20742 USA (e-mail: aandreea@umd.edu).

George J. Pappas is with the Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA 19104 USA (e-mail: pappasg@seas.upenn.edu).

Digital Object Identifier 10.1109/TCNS.2021.3094788

2325-5870 © 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 14:55:33 UTC from IEEE Xplore. Restrictions apply.

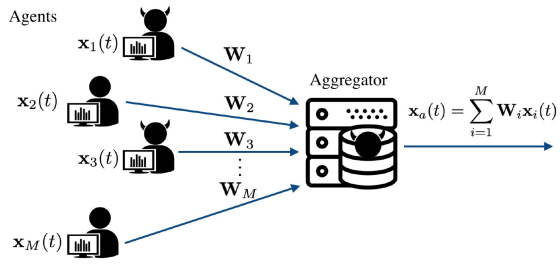


Fig. 1. Diagram of the private weighted sum aggregation. Some of the participants can be corrupted and disclose their private data.

of weighted aggregation, where weights are hidden from all participants. Unlike in the cases where weights are known by the participants in the computation (either by the agents or by the aggregator), this case comes with the challenge that the owner of the weights is not available during the computation, yet the participants are not allowed to learn them. Our second contribution is to efficiently extend these solutions to multidimensional data from scalar data and give valuable optimizations. Our third contribution is to implement and extensively demonstrate the proposed runtime and communication improvements of up to 80% for the problem with hidden weights.

We first review existing solutions for pSA (weights are known by the agents). Most of the previously mentioned literature falls into this category. Second, we describe private weighted sum aggregation with centralized weights (weights are known by the aggregator), which can be solved using functional encryption for inner products. For both, we propose multidimensional extensions. Third, we give a solution for the private weighted sum aggregation with hidden weights (neither agents nor aggregator know the weights) and improve it compared to [17] in terms of security: larger collusion threshold, communication: fewer messages exchanged, and runtime: fewer operations in the multidimensional case. Across the article, we illustrate the changes caused by the different privacy requirements on the data and how the proposed packing benefits only some schemes.

II. Problem Statement

We investigate an aggregation problem of weighted contributions, depicted schematically in Fig. 1. We consider a system with M agents and one aggregator. Each agent $i \in [M]$ has some data $\mathbf{x}_i(t) \in \mathbb{R}^{n_i}$ at time t and the aggregator wants to compute an aggregate of the data in the system $\mathbf{x}_a(t) \in \mathbb{R}^{n_a}$, where $\mathbf{W}_i \in \mathbb{R}^{n_a \times n_i}$ are constant weights designated for the local data of agent i,

$$\mathbf{x}_a(t) = \sum_{i=1}^{M} \mathbf{W}_i \mathbf{x}_i(t). \tag{1}$$

At every time step, each agent $i \in [M]$ has access to its local data $\mathbf{x}_i(t)$, either by direct measurement (e.g., location, energy consumption) or by computation (e.g., gradient of the model, local prediction). We consider three types of privacy requirements for private weighted sum aggregation.

Private Weighted Sum Aggregation with hidden weights: This case requires the strongest privacy guarantees as follows:

a) Agent i should not infer anything about the other agents' local data $\mathbf{x}_j(t)$, $j \in [M] \setminus \{i\}$ or about the aggregator's result $\mathbf{x}_a(t)$ or about the weights $\mathbf{W}_i$, $i \in [M]$, including partial information such as $\mathbf{W}_i\mathbf{x}_i(t)$.
b) The aggregator should only be able to compute $\mathbf{x}_a(t)$ and should not infer anything else about the agents' local data $\mathbf{x}_i(t)$ or the weights $\mathbf{W}_i$, $i \in [M]$, including partial information such as $\mathbf{W}_i\mathbf{x}_i(t)$.

Private Sum Aggregation: In this case, we replace a) by:

a) Agent i knows its corresponding weight $\mathbf{W}_i$ and should not infer anything about the other agents' local data and weights $\mathbf{x}_j(t), \mathbf{W}_j, j \in [M] \setminus \{i\}$, including partial information such as $\mathbf{W}_j\mathbf{x}_j(t)$, or about the aggregator's result $\mathbf{x}_a(t)$.

Private Weighted Sum Aggregation with centralized weights: In this case, we replace b) by:

b) The aggregator knows the weights $\mathbf{W}_i$ and should only be able to compute $\mathbf{x}_a(t)$ and should not infer anything else about the agents' local data $\mathbf{x}_i(t), i \in [M]$, including partial information such as $\mathbf{W}_i\mathbf{x}_i(t)$.

These privacy requirements should hold even under collusion between the aggregator and up to $M-2$ agents: a coalition should not be able to infer the private data of the remaining honest agents. We consider computationally bounded adversaries that are *honest-but-curious*, meaning an adversary wants to learn the private data of the honest agents, without diverging from the established protocol. Such a model is chosen because the aggregator is interested in obtaining the correct result of the computation, and for instance, in applications involving pricing, the agents would be fined if they cheat.

The goal here is to protect the privacy of the inputs and intermediary computations, but reveal the output to the aggregator. As a side note, all the presented algorithms can support differential privacy, in case the output should also be protected.

In describing the schemes in Sections III–V, we focus on scalar data $w_i, x_i, x_a \in \mathbb{Z}_{\geq 0}$, for $i \in [M]$. After illustrating the functionalities, we provide methods for dealing with multidimensional rational data in Sections VII and VIII.

III. Private Sum Aggregation

Private Sum Aggregation, introduced in [29] and [30], enables an untrusted aggregator to compute the sum of the private data contributed by agents, without learning their individual contributions. Improvements in terms of efficiency and functionality of pSA have been proposed in [28], [31]–[35] and the references within. The formal definition of *aggregator obliviousness* that pSA schemes has to satisfy (informally described in Section II) was introduced in [29] and is given as a cryptographic game between an adversary and a challenger, similar to the game, we describe in Appendix C.

When the weights w_i are known to the agents, (1) can be computed privately with a pSA scheme. Specifically, in every time step, denoted by $t \in \mathbb{Z}_{\geq 0}$, each agent $i \in [M]$ holds a private value $x_i(t)$ and w_i. Define $v_i(t) := w_i x_i(t)$. The aggregator wants to compute the aggregate statistics over the private values: $x_a(t) = \sum_{i \in [M]} v_i(t)$.

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 14:55:33 UTC from IEEE Xplore. Restrictions apply.

Let l denote the maximum number of bits of $x_i(t), w_i, \forall i \in [M]$. An assumption that we make for the rest of this article is:

Assumption 1: For each time step t, after discretization, $x_i(t), w_i, w_i x_i(t), x_a(t) < N \ \forall i \in [M]$, i.e., there is no overflow for N specified in each scheme. ◇

The most intuitive pSA scheme involves secret sharing. Each participant will have a secret share of zero for each time step. Each agent will use this share to mask its local data, like a one-time pad—see Appendix B. The aggregator will then sum all the contributions it receives and obtain the desired sum, as the shares of zero will cancel out. The idea of using shares of zero to additively mask private values in aggregation problems was explored, e.g., in [28], [35]–[37].

A pSA scheme should consist of the following algorithms, $\text{pSA}_1 = (\text{Setup}, \text{Enc}, \text{AggrDec})$:

- $\text{Setup}(1^\kappa, M, w_{i\in[M]}, T)$: take as input the security parameter κ, the number of agents M and time period T, and output public parameters prm, secret information for each agent sk_i, $i \in [M]$, and for the aggregator sk_a. This happens as follows: let $N = \max(\kappa, 2l + M)$, then, for each time step $t \in [T]$, generate $M + 1$ shares of zero

$$\sum_{i\in[M]\cup\{a\}} s_i(t) = 0, \quad s_i(t) \in \mathbb{Z}/N\mathbb{Z}.$$

  Set $\text{prm} = (\kappa, M)$, $\text{sk}_i = (s_i(t), w_i)$, $\text{sk}_a = (s_a(t))$.
- $\text{Enc}(\text{prm}, \text{sk}_i, t, x_i(t))$: take as input the public parameters, agent i's secret information, the time step and the local private value. Set $v_i(t) = w_i x_i(t)$ and compute the ciphertext

$$c_i(t) = v_i(t) + s_i(t) \in \mathbb{Z}/N\mathbb{Z}.$$

- $\text{AggrDec}(\text{prm}, \text{sk}_a, t, \{c_i(t)\}_{i\in[M]})$: take as input the public parameters, the aggregator's secret information, the time step and the ciphertexts of the agents for that time step. The aggregator obtains $x_a(t) = \sum_{i\in[M]} v_i(t)$, as follows:

$$x_a(t) = s_a(t) + \sum_{i\in[M]} c_i(t) \bmod N.$$

The correctness of pSA_1 is based on the correct generation of the random shares of zero in Setup, that cancel out after aggregation. Aggregator obliviousness is based on the perfect security of masking the private data in Enc by a one-time pad.

The scheme pSA_1 requires a different set of shares of zero for every time step (otherwise, differences between private contributions at different time steps are leaked). If there exists a trusted dealer, for instance a certificate authority or a system operator that sets up the computation, then this dealer can distribute the required secret shares. Otherwise, the parties distributedly compute these secrets, which can involve elaborate communication, see Section VI. On the other hand, the $\text{pSA}_2 = (\text{Setup}, \text{Enc}, \text{AggrDec})$ scheme from [32], described next, only requires an initial set of shares of zero. This scheme is based on the Paillier cryptosystem [38] (see Appendix A).

- $\text{Setup}(1^\kappa, M, \{w_i\}_{i\in[M]}, T)$: generate p, q to be two equal-size primes and set $N = pq$ with $\gcd(\phi(N), N) = 1$, $\lfloor \log_2 N \rfloor = \kappa$. Define a hash function that acts as a random oracle $H : \mathbb{Z} \to (\mathbb{Z}/N^2\mathbb{Z})^*$. Generate $M + 1$ shares of zero

$$s_a := -\sum_{i\in[M]} s_i, \quad s_i \in (\mathbb{Z}/N^2\mathbb{Z})^*.$$

  Set $\text{prm} = (\kappa, N, H)$, $\text{sk}_i = (s_i, w_i)$, $\text{sk}_a = (s_a)$.
- $\text{Enc}(\text{prm}, \text{sk}_i, t, x_i(t))$: set $v_i(t) = w_i x_i(t)$ and output

$$c_i(t) = (1 + N)^{v_i(t)} H(t)^{s_i} \bmod N^2.$$

- $\text{AggrDec}(\text{prm}, \text{sk}_a, t, \{c_i(t)\}_{i\in[M]})$: compute

$$V(t) = H(t)^{s_a} \prod_{i\in[M]} c_i(t) \bmod N^2$$

$$x_a(t) = (V(t) - 1)/N.$$

The correctness of this scheme follows from the generation of the secret shares and from (18) in Appendix A. The aggregator obliviousness property is proved in [32], based on the security of modular multiplication by a random looking value, and on the decisional composite residuosity assumption (see Appendix A). Furthermore, [32] shows that the security of the scheme is not impacted when the hash function H takes values in $\mathbb{Z}/N^2\mathbb{Z}$, not in $(\mathbb{Z}/N^2\mathbb{Z})^*$.

IV. Private Weighted Sum Aggregation With Centralized Weights

When the aggregator knows the weight corresponding to each of the agents w_i (and they are not identical), but the agents do not know them, we cannot reuse the above pSA schemes. The aggregator is honest-and-curious so we operate under the assumption that the constant weights are not chosen in an adversarial way, e.g., they do not single out one piece of local data. Otherwise, this could be enforced through an external auditor or through zero-knowledge verification tools between the aggregator and the agents.

There are two lines of work that investigate this problem. First, in [39], the authors propose a distributed scenario for aggregation in a graph of agents using a threshold cryptosystem. This implies that after receiving contributions from the agents, the aggregator would ask for help in decrypting the aggregate value. The second line of work removes the extra communication required for decryption. Functional encryption [40] is a generalization of homomorphic encryption and allows a party to compute a functionality over the encrypted data of another party and obtain the desired solution without decryption. One of the few current practical implementations is the functionality of inner products, where one party holds one input and the other party holds the other [41]. Here, we formulate our problem of private weighted sum aggregation with weights known by the aggregator in terms of functional encryption for inner product: $x_a(t) = \langle [w_1, \ldots, w_M], [x_1(t), \ldots, x_M(t)] \rangle$.

The definition of aggregator obliviousness for this case can be written as a cryptographic game formalizing the requirements in Section II. Stronger privacy definitions, from the perspective of functional encryption, can be found in [41].

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 14:55:33 UTC from IEEE Xplore. Restrictions apply.

We modify pSA_2 to get a private weighted sum with centralized weights scheme $\mathrm{pWSAc} = (\mathrm{Setup}, \mathrm{Enc}, \mathrm{AggrDec})$. In Setup, sample M values $s_i \in (\mathbb{Z}/N^2\mathbb{Z})^*$ and set

$$s_a = -\sum_{i\in[M]} w_i s_i. \tag{2}$$

The public key and parameters are the same, but $\mathrm{sk}_i = (s_i)$ and $\mathrm{sk}_a = (\{w_i\}_{i\in[M]}, s_a)$. In the Enc step, compute the ciphertext $c_i(t) = (1+N)^{x_i(t)} \cdot H(t)^{s_i} \bmod N^2$. Finally, in the AggrDec step, the aggregator computes

$$V(t) = H(t)^{s_a} \cdot \prod_{i\in[M]} c_i(t)^{w_i} \bmod N^2$$

and as before, $x_a(t) = (V(t)-1)/N$.

Correctness follows after expanding $V(t)$:

$$V(t) = H(t)^{s_a} \cdot (1+N)^{\sum_{i\in[M]} w_i x_i(t)} \cdot H(t)^{\sum_{i\in[M]} w_i s_i}.$$

Using $s_a = -\sum_{i\in[M]} w_i s_i$, we obtain that $(V(t)-1)/N = \sum_{i\in[M]} w_i x_i(t) = x_a(t)$, as needed. Aggregator obliviousness follows from the proof in [32], where the secret of the aggregator is now $s_a = -\sum_{i\in[M]} w_i s_i$ instead of $-\sum_{i\in[M]} s_i$, and the aggregator raises the ciphertexts of the participants to the respective power w_i.

Notice that the keys are independent of the time period T. The Setup step can be performed by a dealer that does not need to know the weights of the aggregator. The dealer generates M random secrets s_i and sends one to each agent. The aggregator generates the public and secret key of an additively homomorphic encryption scheme, e.g., Paillier [38], encrypts the weights w_i, for $i \in [M]$ and sends them to the dealer. Then, the dealer computes (2) as $\mathrm{E}(s_a) = \prod_{i\in[M]} \mathrm{E}(w_i)^{-s_i}$, and sends it to the aggregator, which then simply decrypts s_a.

V. Private Weighted Sum Aggregation With Hidden Weights

A private weighted sum aggregation scheme for weights unknown to all participants is composed of algorithms $\mathrm{pWSAh} = (\mathrm{Setup}, \mathrm{InitW}, \mathrm{Enc}, \mathrm{AggrDec})$. The formal security definition of aggregator obliviousness is given in Definition C.1 in Appendix C as a cryptographic game. This game mimics the real execution of the scheme, but with a more powerful adversary that can choose both the local data of the corrupted participants and the local data of uncorrupted participants. If even in this case, the adversary is not able to break the privacy of the scheme, then the scheme is private also when multiple participants collude and share their private information, but cannot set the private data of the honest participants.

We first note that, in the context of cooperative control, local control laws of the form (1) were considered in [16]–[18]. Specifically, [16] introduces a private computation and exchange of the "input portions" $\mathbf{v}_i(t) := \mathbf{W}_i\mathbf{x}_i(t)$, that reveal neither the exact local state $\mathbf{x}_i$ nor the local controller matrix $\mathbf{W}_i$ to the aggregator, but can at least reveal the relative rate of decrease/increase of some signals of the agents over multiple time steps. More details about the information leak can be found in [17], where a solution to the pWSAh problem that achieves aggregator obliviousness is proposed. The solution in [17] required generating fresh secrets at every time step and proposed an online decentralized method that reduced the collusion threshold between participants. Finally, Alexandru and Pappas [18] proposed a theoretical solution that addressed the two issues of [17]: it reduced the number of generated secrets, kept the collusion threshold at $M-1$ corrupted participants and reduced the number of messages exchanged between the agents and aggregator. This came at the cost of using a lattice-based homomorphic encryption scheme and augmented learning with error ciphertexts, which can be larger than Paillier ciphertexts and might require more expensive operations.

Our current work extends the results in [17] (and avoids the more complex cryptographic tools in [18]) in the following way: we provide an online decentralized method of distributing the secret shares of zero among the agents without reducing the collusion threshold, and we propose a more compact and efficient private weighted sum aggregation scheme by packing multiple values in one homomorphic Paillier ciphertext.

In pWSAh, the weight w_i should be private from all participants, so one solution is to encrypt it. Then, agent i has to be able to send an encryption of the masked product $w_i x_i(t)$ to the aggregator, and the latter has to be able to compute and decrypt the result. This suggests the following outline :

- w_i should be encrypted with an additively homomorphic encryption that the aggregator knows how to decrypt;
- the layer of encryption introduced in Enc should be compatible with the inner additively homomorphic layer;
- the aggregator should not be able to decrypt the individual contributions it receives from the agents, despite having the secret key of the homomorphic encryption scheme.

To achieve the solution, we use a combination of the two schemes described in Section III. For the outer layer of encryption, we use one-time pads as in pSA_1, which are compatible with the additively homomorphic property. For the inner layer of encryption, we use an asymmetric additive homomorphic encryption scheme. We instantiate it with the Paillier cryptosystem [38], described in Appendix A.

Hence, the steps of the algorithms in pWSAh are as follows:

- $\mathrm{Setup}(1^\kappa, M, T)$: given the security parameter κ, generate two equal-size prime numbers p, q, and set $N = pq$ so that $\lfloor \log_2 N \rfloor = \kappa$ and $\gcd(\phi(N), N) = 1$. Set the Paillier keypair $\mathfrak{pk} = (N), \mathfrak{sk} = (\phi(N), \phi(N)^{-1} \bmod N)$. For every $t \in [T]$, generate $M+1$ shares of zero
$$s_a(t) := -\sum_{i\in[M]} s_i(t), \quad s_i(t) \in (\mathbb{Z}/N\mathbb{Z})^*.$$
Finally, set $\mathrm{prm} = (\kappa, \mathfrak{pk})$, $\mathrm{sk}_i = (s_i(t \in [T]))$ and $\mathrm{sk}_a = (\mathfrak{sk}, s_a(t \in [T]))$.
- $\mathrm{InitW}(\mathrm{prm}, M, \{w_i\}_{i\in[M]})$: given the public key of the Paillier scheme pk, encrypt w_i for $i \in [M]$ and return $\mathrm{sw}_i = \mathrm{E}(w_i) = (1+N)^{w_i} r^N \bmod N^2$, for r randomly sampled from $\mathbb{Z}/N\mathbb{Z}$ and such that $\gcd(r, N) = 1$.
- $\mathrm{Enc}(\mathrm{prm}, \mathrm{sw}_i, \mathrm{sk}_i, t, x_i(t))$: for $x_i(t) \in \mathbb{Z}/N\mathbb{Z}$, set $c_i(t) = \mathrm{E}(w_i)^{x_i(t)} \cdot \mathrm{E}(s_i(t)) = \mathrm{E}(w_i x_i(t) + s_i(t))$.
- $\mathrm{AggrDec}(\mathrm{prm}, \mathrm{sk}_a, t, \{c_i(t)\}_{i\in[M]})$: compute $V(t) = \prod_{i\in[M]} c_i(t) \bmod N^2$ and then set
$$x_a(t) = (\mathrm{D}(V(t)) + s_a(t)) \bmod N.$$

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 14:55:33 UTC from IEEE Xplore. Restrictions apply.

Correctness: $\mathrm{D}(V(t)) = \sum_{i\in[M]} w_i x_i(t) + s_i(t)$ follows from the correct execution of Paillier operations in Enc. Then, $\mathrm{D}(V(t)) + s_a(t) = \sum_{i\in[M]} w_i x_i(t) \bmod N = x_a(t)$ from the generation of shares of zero.

Theorem 1: The pWSAh scheme achieves weighted sum aggregator obliviousness w.r.t. Definition C.1. ◇

The proof is given in Appendix D.

Remark 1: Unlike in pSA_2, in pWSAh the aggregator has to know the secret key of the cryptosystem that encrypts the weights. If we would use pSA_2, which has a single share of zero per agent for all time steps, an adversary that corrupts the aggregator and selects equal contributions at different time steps for an agent in the pWSAO game (described in Appendix C) could learn that agent's secret share. ◇

The above scheme is appealing due to its simplicity, but involves demanding communication, because secret shares of zero are required at every time step t for every participant. The Setup is executed by an incorruptible trusted third-party, called dealer, which cannot be online at every time step to distribute the shares. A more reasonable assumption is that, prior to the online computations, the dealer computes the shares for T time steps and sends them to the agents, who have to store them. Alternatively, we offer below a solution to generate the secret shares of zero in a distributed way, without the need of a trusted third-party.

VI. Decentralized Generation of Zero Shares

A. One Communication Round, Lower Threshold

In [17], we offered a solution with one round of communication but lower collusion threshold. This solution is appealing in the case where the agents are connected by a dense graph. Specifically, at each time step, each agent would generate and send shares to the agents in its neighborhood (including the aggregator), then sum up the shares it received from its neighbors. This guarantees that all participants will hold a share of zero. However, if the communication graph between the agents is sparse, the collusion threshold drops from $M-1$ participants to the minimum number of neighbors that an agents has.

B. Two Communication Rounds, Same Threshold

If new communication links can be enforced between the agents, we can create dummy connections such that each agent reaches a desired vertex degree, thus increasing the collusion threshold. This keeps the same number of communication rounds as before, but the cost of adding new communication links might be too expensive. For instance, if connections are based on proximity, such a solution is not reasonable.

Alternatively, if we allow each agent to obtain a valid share of zero in two rounds, we can retrieve the initial collusion threshold of $M-1$ participants. Instead of sending the shares to each other, the agents use the aggregator as an intermediate relay to get to the agents that they are not directly connected to. Specifically, each agent $i \in [M]$ will generate $M+1$ shares and encrypt them with a key known the agent $l \in ([M] \setminus i) \cup a$ (with, e.g., a symmetric encryption like AES). The aggregator will forward the corresponding shares to its neighbors $l \neq i$.

1) At time $t-2$, each agent and the aggregator $i \in [M] \cup a$ creates shares of zero $\sigma_{il}(t) \in (\mathbb{Z}/N\mathbb{Z})^*$ for itself and for the rest of the agents
$$\sum_{l\in[M]\cup a} \sigma_{il}(t) = 0. \tag{3}$$
It encrypts them with a key known to agent $l \in [M] \cup a \setminus i$ and sends $\mathrm{AES}(\mathrm{key}_l, \sigma_{il}(t))$ to the aggregator.
2) At time $t-1$, the aggregator batches the M shares for agent $i \in [M]$ and sends them.
3) At time t, each agent $l \in [M] \cup a$ sums its own share and the shares it received and decrypted from the aggregator:
$$s_l(t) := \sum_{i\in[M]\cup a} \sigma_{il}(t) \bmod N. \tag{4}$$

In order to reduce the load on the aggregator, an agent i can communicate directly to agents $l \in \mathcal{N}_i \cap [M]$, where $\mathcal{N}_i$ is the set of neighbors of agent i and only send the encrypted shares to the aggregator for the agents $l \notin \mathcal{N}_i \cap [M]$.

VII. Packed Paillier Scheme

Assume we have a vector $\mathbf{y} = [\mathbf{y}^{[1]}, \mathbf{y}^{[2]}, \ldots, \mathbf{y}^{[m]}]$, with $\mathbf{y}^{[i]} \in [0,\, 2^l) \cap \mathbb{Z}_{\geq 0}$. We can take advantage of the fact that $N >> 2^l$, where N is the Paillier modulus, by packing m items of l bits into a plaintext in $\mathbb{Z}_N$ in the following way:

$$p_y = \sum_{i=1}^{m} \mathbf{y}^{[i]} 2^{l(i-1)} = [\mathbf{y}^{[1]} | \mathbf{y}^{[2]} | \ldots | \mathbf{y}^{[m]}].$$

Here, we depict the least significant bits on the left, to show the elements in the order that they appear in the vector. If we need to perform additional operations on p_y after packing, we have to make sure we retrieve the correct elements. Hence, we need to take into account possible overflows from one "slot" of the ciphertext the another. We do this by padding with enough zeroes, where $\delta > l$ and will be determined based on the computations performed on p_y afterward:

$$p_y = \sum_{i=1}^{m} \mathbf{y}^{[i]} 2^{\delta(i-1)} = [\underbrace{\mathbf{y}^{[1]}}_{l}\underbrace{0\ldots0}_{\delta-l} | \underbrace{\mathbf{y}^{[2]}}_{l}\underbrace{0\ldots0}_{\delta-l} | \ldots | \underbrace{\mathbf{y}^{[m]}}_{l}\underbrace{0\ldots0}_{\delta-l}]. \tag{5}$$

Note that we can perform (5) as long as $m\delta < N$.

In (5), we require positive integers. For a value $y \in \mathbb{Q}_{l_i,l_f}$, we first construct an integer $\overline{y}$, where $l := l_i + l_f$, and then obtain a positive integer $\tilde{y}$, for $\gamma > l$ that we will specify later,

$$\overline{y} := y2^{l_f} \Rightarrow \bar{y} \in [-2^{l-1}, 2^{l-1}) \cap \mathbb{Z} \tag{6}$$

$$\tilde{y} := \overline{y} + 2^{\gamma} \Rightarrow \tilde{y} \in [0, 2^{\gamma+1}) \cap \mathbb{Z}_{\geq 0}. \tag{7}$$

In Sections VIII-B and VIII-C1, where we do not use packing, instead of (7) we use (assuming $2^{l-1} < N/2$)

$$\tilde{y} := \begin{cases} \bar{y} & \text{if } \bar{y} \geq 0 \\ \bar{y} + N & \text{if } \bar{y} < 0 \end{cases} \Rightarrow \tilde{y} \in \mathbb{Z}/N\mathbb{Z}. \tag{8}$$

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 14:55:33 UTC from IEEE Xplore. Restrictions apply.

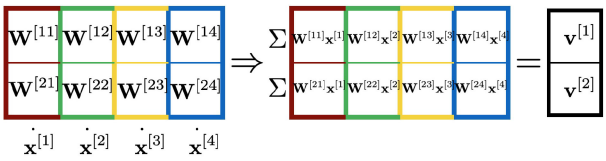


Fig. 2. Diagram of column-packed matrix-vector multiplication. Theentries with the same outer coloring are packed in the same ciphertext.

Batching multiple entries into one Paillier ciphertext was first proposed in [42]. Notice that after packing multiple plaintexts into one Paillier ciphertext as in (5), we can still perform the homomorphisms corresponding to element-wise addition and scalar multiplication. In the following, we investigate a more efficient way to compute an encrypted matrix-plaintext vector multiplication, by using only packing, element-wise addition and scalar multiplication. Fig. 2 depicts this method. For a matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$, denote the jth column by $\mathbf{w}^j$, for $j \in [n]$. Then, in order to obtain the product $\mathbf{v} := \mathbf{W}\mathbf{x}$, we multiply each column $\mathbf{w}^j$ by the corresponding element in the vector $\mathbf{x}^{[j]}$ and then sum over all the obtained vectors

$$\mathbf{v} = \sum_{j=1}^{n} \mathbf{w}^j \mathbf{x}^{[j]}. \tag{9}$$

VIII. Schemes for Multidimensional Data

A. Multidimensional pSA

In the case of pSA_1, the messages have small sizes and communication is less of a problem even for multidimensional data. However, in the case of pSA_2, messages (ciphertexts) are larger and we propose a better method than sending a different message for each element of the resulting vector, by batching the elements in a single ciphertext. The aggregator wants to obtain $\mathbf{x}_a(t) = \sum_{i\in[M]} \mathbf{W}_i\mathbf{x}_i(t) =: \sum_{i\in[M]} \mathbf{v}_i(t)$. In pSA_2, the dealer generates the secret shares the same way as previously, but each agent $i \in [M]$ computes $\mathbf{v}_i^{[j]}(t)$ and uses (6) and (7) to obtain $\tilde{\mathbf{v}}_i^{[j]}(t)$, then computes $p_i(t) = \sum_{j\in[n_i]} \tilde{\mathbf{v}}_i^{[j]}(t)2^{\delta(j-1)}$, $c_i(t) = (1+N)^{p_i(t)}H(t)^{s_i} \bmod N^2$. The aggregator computes $V(t)$ as before, and retrieves

$$\tilde{\mathbf{x}}_a^{[k]} = V(t)//2^{(n_a-k)\delta} \bmod 2^{(k-1)\delta}, \quad k \in \{2,\dots,n_a-1\}$$

and $\tilde{\mathbf{x}}_a^{[1]} = V(t) \bmod 2^{\delta}$, $\tilde{\mathbf{x}}_a^{[n_a]} = V(t)//2^{(n_a-1)\delta}$, where by $//$ we mean the quotient operation. From the elements of $\tilde{\mathbf{x}}_a(t)$, the aggregator needs to subtract $2^{\gamma}M$ and divide by 2^{2l_f} each element, in order to obtain $\mathbf{x}_a(t)$.

Choosing $\gamma = 2l+1$ and $\delta = 2l+2+\lceil \log_2 M\rceil$ ensures the correctness of the decryption, as no overflow occurs.

B. Multidimensional pWSAc

Here, we cannot use packing to reduce the number of ciphertexts because we would require rotations and element-wise multiplications, which cannot be performed on packed Paillier ciphertexts. Compared to the pWSAh scheme, pWSAc has the advantage that only one set of secret shares are needed for all time steps, hence, communication due to secret generation and sharing only happens once.

In the multidimensional case, the participants perform the operations in Section IV for every element, as described next. First, they prepare their data using (6) and (8) element-wise. In Setup, n_aM secrets $\mathbf{s}_i \in ((\mathbb{Z}/N\mathbb{Z})^*)^{n_a}$ are generated and

$$\mathbf{s}_a^{[k]} = -\sum_{i\in[M]}\sum_{j\in[n_i]} \mathbf{W}_i^{[kj]}\mathbf{s}_i^{[j]}, \quad k \in [n_a].$$

In Enc, each agent $i \in [M]$ constructs n_a ciphertexts

$$\mathbf{c}_i^{[j]}(t) = (1+\tilde{\mathbf{x}}_i^{[j]}(t)N)\cdot H(t)^{\mathbf{s}_i^{[j]}} \bmod N^2, \quad j \in [n_i].$$

Finally, in AggrDec, the aggregator computes for $k \in [n_a]$

$$\mathbf{V}^{[k]}(t) = H(t)^{\mathbf{s}_a^{[k]}} \cdot \prod_{i\in[M]}\prod_{j\in[n_i]} \mathbf{c}_i^{[j]}(t)^{\tilde{\mathbf{W}}_i^{[kj]}} \bmod N^2$$

and $\tilde{\mathbf{x}}_a^{[k]}(t) = (\mathbf{V}^{[k]}(t)-1)/N$. The aggregator retrieves the elements of $\mathbf{x}_a(t)$ from $\tilde{\mathbf{x}}_a(t)$ by subtracting N from the elements greater than $N/2$ and dividing all of them by 2^{2l_f}.

C. Multidimensional pWSAh

We consider values on l bits, with $\log_2 N >> l$, for N ensuring semantic security of the Paillier scheme. Sampling a random value from a large $\mathbb{Z}/N\mathbb{Z}$ is expensive, but also redundant, since it masks a much smaller message. To this end, we prefer to sample $s_i(t) \in (0, 2^{\lambda+2l}), \forall i \in [M]$, for λ the statistical security parameter and to set $s_a(t) := -\sum_{i\in[M]} s_i(t)$ in pWSAh. Masking by $s_i(t)$ will guarantee λ-statistical security rather than perfect security, see Appendix B. From here on, we use this more efficient approach.

1) Naive Multidimensional Scheme: This solution was proposed in [17] and performs each step in Section V element-wise. In Setup, for every $t \in [T]$, $M \cdot n_a$ shares of zero $\mathbf{s}_i^{[k]}(t) \in (0, 2^{\lambda+2l})$ are generated for $i \in [M], k \in [n_a]$

$$\mathbf{s}_a^{[k]}(t) = -\sum_{i\in[M]}\sum_{k\in[n_a]} \mathbf{s}_i^{[k]}(t).$$

In InitW, the weights $\mathbf{W}_i$, $i \in [M]$ are processed as in (6) and (8) and encrypted element-wise: $\mathrm{E}(\mathbf{W}_i^{[kj]}) = (1+N)^{\tilde{\mathbf{W}}_i^{[kj]}} r^N \bmod N^2$, for r randomly sampled from $\mathbb{Z}/N\mathbb{Z}$ and satisfying $\gcd(r, N) = 1$. In Enc, each agent $i \in [M]$ computes

$$\mathbf{c}_i^{[k]}(t) = \prod_{j\in[n_i]} \mathrm{E}(\tilde{\mathbf{W}}_i^{[kj]})^{\tilde{\mathbf{x}}_i^{[j]}(t)} \cdot \mathrm{E}(\mathbf{s}_i^{[k]}(t)), \forall k \in [n_a]$$

equivalent to $\mathbf{c}_i^{[k]}(t) = \mathrm{E}(\sum_{j\in[n_i]} \mathbf{W}_i^{[kj]}\mathbf{x}_i^{[j]}(t) + \mathbf{s}_i^{[k]}(t))$. Finally, in AggrDec, the aggregator computes, for $k \in [n_a]$

$$\mathbf{V}^{[k]}(t) = \prod_{i\in[M]} \mathbf{c}_i^{[k]}(t) \bmod N^2$$

$$\tilde{\mathbf{x}}_a^{[k]}(t) = \mathrm{D}(\mathbf{V}^{[k]}(t)) + \mathbf{s}_a^{[k]}(t).$$

2) Packed Multidimensional Scheme: We reduce the number of ciphertexts and the corresponding number of operations by using packing and the more efficient encrypted matrix-plaintext vector multiplication described in Section VII.

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 14:55:33 UTC from IEEE Xplore. Restrictions apply.

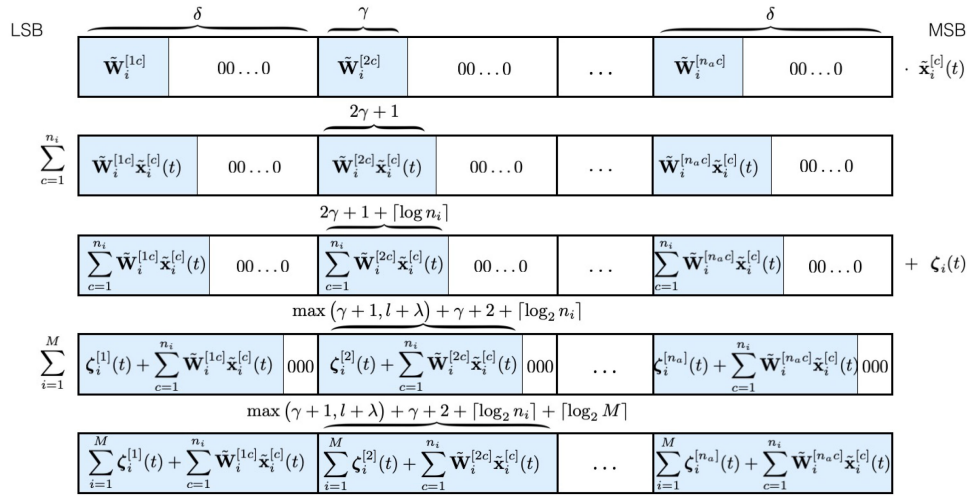


Fig. 3. Operations performed on the packed columns and the corresponding number of bits of the result, where on the third line $\boldsymbol{\zeta}_i(t) = [\boldsymbol{\zeta}_i^{[1]}(t)\,0\ldots0\,\boldsymbol{\zeta}_i^{[2]}(t)\ldots\boldsymbol{\zeta}_i^{[n_a]}(t)\,0\ldots0]$ and $\boldsymbol{\zeta}_i^{[k]}(t) = \mathbf{s}_i^{[k]}(t) + 2^{\gamma}\mathbf{z}_i^{[k]}(t)$, for $k \in [n_a]$.

Assume at the moment that we can pack at least n_a values in one ciphertext. The steps we take are as follows: 1) Preprocess the values to be positive and integer; 2) Pack and encrypt the columns of the matrix $\mathbf{W}_i$ and obtain n_i ciphertexts; 3) Perform a scalar multiplication of one encrypted column c with the scalar $\mathbf{x}_i^{[c]}(t)$; 4) Sum the n_i ciphertexts to get the encryption of $\mathbf{W}_i\mathbf{x}_i(t)$; 5) Add the share of zero and mask the intermediate results; 6) Sum the M ciphertexts to obtain the encryption of $\sum_{i\in[M]}\mathbf{W}_i\mathbf{x}_i(t)$; 7) Decrypt, unmask and unpack the result.

Next, we detail these steps and depict the bit gains due to the operations performed on the packed columns in Fig. 3.

1) Prepare the values to be packed as per (6) and (7), with γ to be determined later.

2) Construct the packed plaintext $p_i^c = \sum_{k=1}^{n_a}\tilde{\mathbf{W}}_i^{[kc]}2^{(k-1)\delta}$; encrypt it in $\mathrm{E}(\mathbf{W}_i^c)$.

3) Multiply the encrypted column $\mathrm{E}(\mathbf{W}_i^c)$ by a preprocessed scalar $\tilde{\mathbf{x}}_i^{[c]}(t)$. One slot of the ciphertext will now contain (10) and will be represented on $2\gamma+2$ bits

$$(\overline{\mathbf{W}}_i^{[kc]} + 2^{\gamma})(\overline{\mathbf{x}}_i^{[c]}(t) + 2^{\gamma}) = \overline{\mathbf{W}}_i^{[kc]}\overline{\mathbf{x}}_i^{[c]}(t) + 2^{2\gamma} + 2^{\gamma}(\overline{\mathbf{W}}_i^{[kc]} + \overline{\mathbf{x}}_i^{[c]}(t)). \tag{10}$$

From (10), one can retrieve the desired result $\overline{\mathbf{W}}_i^{[kc]}\overline{\mathbf{x}}_i^{[c]}(t)$ by taking the lefthand side modulo 2^{γ}. But one can also obtain

$$\overline{\mathbf{W}}_i^{[kc]} + \overline{\mathbf{x}}_i^{[c]}(t) = \lfloor(\tilde{\mathbf{W}}_i^{[kc]}\tilde{\mathbf{x}}_i^{[c]}(t) - 2^{2\gamma})//2^{\gamma}\rfloor \tag{11}$$

which can reveal intermediate information to the decryptor. In order to avoid this information leakage, we need to artificially add some noise $\mathbf{z}_i^{[kc]}(t)$ that still allows retrieving $\overline{\mathbf{W}}_i^{[kc]}\overline{\mathbf{x}}_i^{[c]}(t)$. It is more efficient to add this noise in step 5), after performing the sum over n_i.

4) Sum the n_i ciphertexts to output a ciphertext that contains the vector result of the matrix–vector multiplication product.

5) For each slot $k \in [n_a]$, an agent $i \in [M]$ selects $\mathbf{z}_i^{[k]}(t) \in (0, 2^{l+1+\lambda+\lceil\log_2 n_i\rceil})$, such that a statistical security of λ bits is guaranteed for the private value $\sum_{c=1}^{n_i}\overline{\mathbf{W}}_i^{[kc]} + \overline{\mathbf{x}}_i^{[c]}(t)$. Then, each agent constructs its ciphertext $c_i(t)$ by adding the secret shares of zero such that the remaining private value $\sum_{c=1}^{n_i}\overline{\mathbf{W}}_i^{[kc]}\overline{\mathbf{x}}_i^{[c]}(t)$ is concealed

$$\begin{aligned} c_i(t) &:= \mathbf{s}_i^{[k]}(t) + 2^{\gamma}\mathbf{z}_i^{[k]}(t) + \sum_{c=1}^{n_i}\tilde{\mathbf{W}}_i^{[kc]}\tilde{\mathbf{x}}_i^{[c]}(t) \\ &\overset{(10)}{=} n_i 2^{2\gamma} + \mathbf{s}_i^{[k]}(t) + \sum_{c=1}^{n_i}\overline{\mathbf{W}}_i^{[kc]}\overline{\mathbf{x}}_i^{[c]}(t) \\ &\quad + 2^{\gamma}\left(\mathbf{z}_i^{[k]}(t) + \sum_{c=1}^{n_i}\overline{\mathbf{W}}_i^{[kc]} + \overline{\mathbf{x}}_i^{[c]}(t)\right). \end{aligned} \tag{12}$$

Due to the masking with $2^{\gamma}z_i^{[k]}(t)$, we can reduce the size of the mask $\mathbf{s}_i^{[k]}(t)$. More specifically, $\mathbf{s}_i^{[k]}(t)$ can be sampled uniformly at random from $(0, 2^{\gamma})$, because it acts like a one-time pad (perfect secrecy) on $\sum_{c=1}^{n_i}\overline{\mathbf{W}}_i^{[kc]}\overline{\mathbf{x}}_i^{[c]}(t)$ once the decryptor takes equation (12) modulo 2^{γ}. The whole quantity $\mathbf{s}_i^{[k]}(t) + 2^{\gamma}\mathbf{z}_i^{[k]}(t)$ is used for masking, but we only need to ensure that $\sum_{i\in[M]\cup a}\mathbf{s}_i^{[k]}(t) = 0$.

6) The aggregator obtains $c(t)$ by taking the product of all ciphertexts $c_i(t)$ it received.

7) The aggregator decrypts the ciphertext $c(t)$, adds its own secret share of zero $\mathbf{s}_a(t)$. It then retrieves the n_a elements of $\tilde{\mathbf{x}}_a(t)$ by recursively taking the quotient and rest by 2^{δ}, and from each resulting element, it obtains the elements of $\mathbf{x}_a(t)$ by taking modulo 2^{γ} and dividing by 2^{2l_f}.

We now compute the number of bits and corresponding padding one slot can have such that no overflow occurs during the private weighted sum aggregation. We assume that the values of n_i, for $i \in [M]$ are similar and define $n := \max_{i\in[M]} n_i$. The

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 14:55:33 UTC from IEEE Xplore. Restrictions apply.

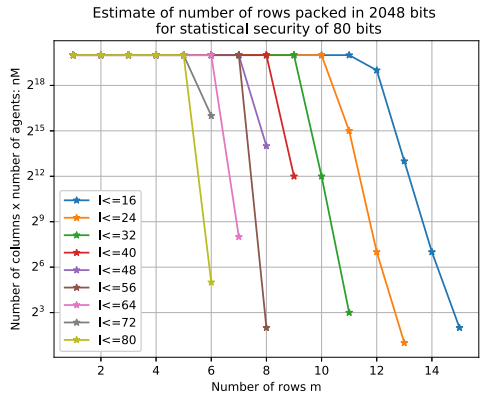


Fig. 4. Number of rows m that can be packed in a plaintext of $N = 2048$ b, as a function of the number of columns n, number of agents M, size of message l and statistical security of $\lambda = 80$ b.

value that is packed in slot $k \in [n_a]$ after step 6) is

$$\sum_{i=1}^{M}\left(\mathbf{s}_i^{[k]}(t) + 2^{\gamma}\mathbf{z}_i^{[k]}(t) + \sum_{c=1}^{n_i}\tilde{\mathbf{W}}_i^{[kc]}\tilde{\mathbf{x}}_i^{[c]}(t)\right) < 2^{\delta}$$

from which we obtain that $\delta > \max(\gamma+1, l+\lambda) + \lceil\log_2 n\rceil + \lceil\log_2 M\rceil + \gamma + 2$. For the correct retrieval of the desired result, we require that

$$\sum_{i=1}^{M}\sum_{c=1}^{n_i}\overline{\mathbf{W}}_i^{[kc]}\overline{\mathbf{x}}_i^{[c]}(t) < 2^{\gamma}. \tag{13}$$

From (13), we choose $\gamma = 2l + 1 + \lceil\log_2 n\rceil + \lceil\log_2 M\rceil$ and

$$\begin{aligned}\delta = \ &\max(l + 2 + \lceil\log_2 n\rceil + \lceil\log_2 M\rceil, \lambda) + \\ &3l + 4 + 2(\lceil\log_2 n\rceil + \lceil\log_2 M\rceil).\end{aligned} \tag{14}$$

Fig. 4 indicates possible values for the number of rows, columns, and agents depending on the plaintext size and statistical security. Denote the maximum number of values we can pack by $m < N/\delta$. If $n_a > m$, we split the columns into $\lceil n_a/m\rceil$ Paillier ciphertexts and follow the same operations as before, and concatenate the resulting vectors after decryption.

Let $\text{pWSAh}^* = (\text{Setup}, \text{InitW}, \text{Enc}, \text{AggrDec})$ be a packed multidimensional scheme for private weighted sum aggregation with hidden weights, where steps 1) and 2) are performed by the dealer as part of InitW and the shares of zero for step 5) are generated as part of Setup, steps 1), 3)–5) are performed by each agent $i \in [M]$ in Enc and steps 6) and 7) are performed by the aggregator in AggrDec.

Theorem 2: The packed multidimensional scheme pWSAh^* is correct and achieves aggregator obliviousness.

Proof: The correctness follows from the appropriate padding and packing to avoid overflow, as stated in (14). The aggregator obliviousness proof follows from Theorem 1, along with the intermediate values masking from (12). ■

D. Comparison Between Naive and Packed Method

In the naive version of the multidimensional pWSAh, each agent receives $n_a n_i$ ciphertexts for $\mathbf{W}_i$ at the initialization of the protocol, then computes $n_a n_i$ ciphertext—scalar multiplications (modular exponentiation), $n_a(n_i - 1)$ ciphertext additions (modular multiplications) and sends to the aggregator n_a ciphertexts. In the decentralized way of generating shares, each agent will have to send out $(2l + \lambda)n_a$ bits of randomness to each of the M neighbors per time step. Denote by m the number of elements we can pack in a Paillier ciphertext. In the packed version pWSAh^*, each agent receives $\lceil n_a/m\rceil n_i$ ciphertexts for $\mathbf{W}_i$ at the initialization of the protocol, then computes $\lceil n_a/m\rceil n_i$ ciphertext—scalar multiplications, $\lceil n_a/m\rceil(n_i - 1)$ ciphertext additions and sends to the aggregator $\lceil n_a/m\rceil$ ciphertexts. In the decentralized way of generating shares, each agent will have to send out $(2l + 1 + \lceil n\rceil + \lceil M\rceil)\lceil n_a/m\rceil$ bits of randomness to each of the M neighbors per time step.

IX. Case Study: Private Distributed Control

For illustration purposes, we consider a distributed linear control scheme for linear dynamics of M agents in a network

$$\mathbf{x}_i(t+1) = \mathbf{A}_i\mathbf{x}_i(t) + \mathbf{B}_i\mathbf{u}_i(t), \quad \mathbf{x}_i(0) = \mathbf{x}_{i,0} \tag{15}$$

with $\mathbf{x}_i \in \mathbb{R}^{n_i}$ and $\mathbf{u}_i \in \mathbb{R}^{m_i}$, for every $i \in [M]$. The agents are part of an undirected connected communication graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with vertex set $\mathcal{V} = [M]$ and edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. An edge $(i, j) \in \mathcal{E}$ specifies that agent i can communicate with agent j, i.e., agent i and agent j are neighbors.

We can use the local control laws to stabilize the systems

$$\mathbf{u}_i(t) = \mathbf{K}_{ii}\mathbf{x}_i(t) + \sum_{j\in\mathcal{N}_i}\mathbf{K}_{ij}\mathbf{x}_j(t) \tag{16}$$

where $\mathcal{N}_i := \{j \in \mathcal{V} | (i, j) \in \mathcal{E}\}$ represents the set of neighbors of agent i. The local control laws (16) result from the design of a centralized linear controller of the form

$$\begin{pmatrix}\mathbf{u}_1(t)\\ \vdots \\ \mathbf{u}_M(t)\end{pmatrix} = \begin{pmatrix}\mathbf{K}_{11} & \dots & \mathbf{K}_{1M}\\ \vdots & \ddots & \vdots \\ \mathbf{K}_{M1} & \dots & \mathbf{K}_{MM}\end{pmatrix}\begin{pmatrix}\mathbf{x}_1(t)\\ \vdots \\ \mathbf{x}_M(t)\end{pmatrix} \tag{17}$$

that accounts for the structural constraints of the communication graph by requiring $\mathbf{K}_{ij} = \mathbf{0}$ if $j \notin \mathcal{N}_i \cup \{i\}$, see, e.g., [3].

In this illustrative example, there is a system operator who designs the control feedback weights $\mathbf{K}$, based on the structural constraints of the communication graph, as well as on the model of the distributed system and on proprietary knowledge and costs. The system operator wants to keep this information private from potential adversaries that can capture or corrupt the agents, hence encrypts the gains $\mathbf{K}_{ij}$ for each agent and acts as a dealer in the pWSAh^* protocol. Each agent i is the aggregator of the contributions of its neighbors $\mathbf{K}_{ij}\mathbf{x}_j(t)$, and desires to keep its own local data private from potential eavesdroppers and corrupted neighbors.

Specifically, consider a network of 50 agents, with each agent having six local states and six local control inputs. We simulate pWSAh^* for various values of the average node degree in the network, obtained by varying the probability of drawing edges between agents. Simulations were run in Python 3 on a 2.2 GHz Intel Core i7 processor. As usual when operating with encryption schemes defined on groups of modular residues, we need to use a fixed-point representation scheme in order to quantize the real values and encode them into integers. The effect of this quantization on the cooperative control scheme is described

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 14:55:33 UTC from IEEE Xplore. Restrictions apply.

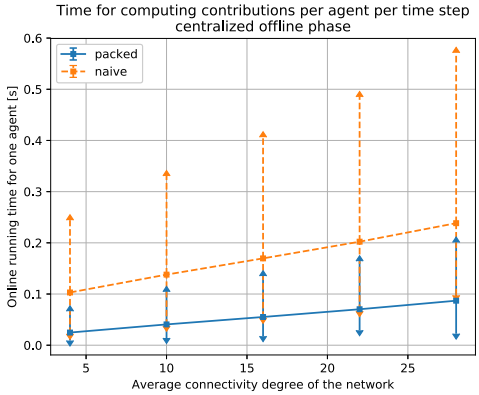


Fig. 5. Average running times for the pWSAh* scheme with the steps described in Section V in a network of 50 agents.

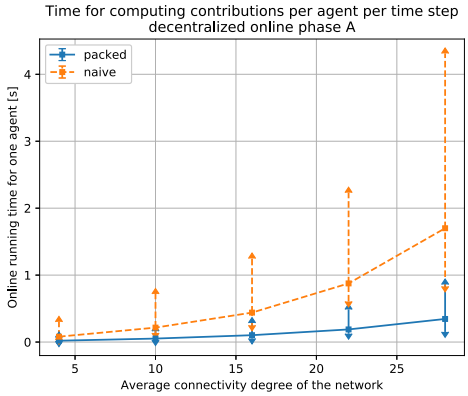


Fig. 6. Average running times for the pWSAh* scheme with the steps described in Section VI-A in a network of 50 agents.

in [16]. In the simulations, we choose the message representation to be on $l = 32$ bits: 16 integer bits and 16 fractional bits, the statistical security size to be 80 b and the Paillier moduli for each agent to have 2048 b. With these chosen values, all six elements of the local contributions can be encoded into a single Paillier ciphertext in pWSAh*.

We present the simulation results for the solutions described in Sections V, VI-A, and VI-B, in both naive implementation (see Section VIII-C1) and using packing (see Section VIII-C2). The running times are averaged over 50 instances and represent the total time it takes for an agent at a time step to generate and distribute the secret shares for the computation for itself and its neighbors *and* to aggregate the contributions of its neighbors *and* to compute and send out its own contribution to its neighbors. The shares are locally encrypted with an AES cipher with 128-bit key—for the offline centralized phase, each agent has an AES key with the dealer, and each pair of neighbors has their own AES key for the online decentralized phase. Figs. 5, 6, 7 and 9 show the times for an agent with the average connectivity degree, respectively, the minimum and maximum connectivity degree (represented by the arrows).

Fig. 5 compares the running times for the local computation at each agent in a time step using the scheme with centralized offline generation of the secret shares, between naive, and packed encryption. This method has the smallest online running time for agents, but the largest offline time for the dealer that generates

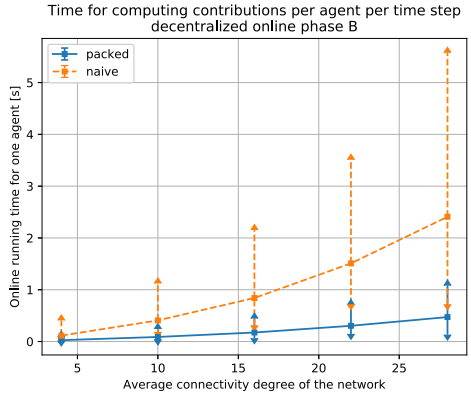


Fig. 7. Average running times for the pWSAh* scheme with the steps described in Section VI-B in a network of 50 agents.

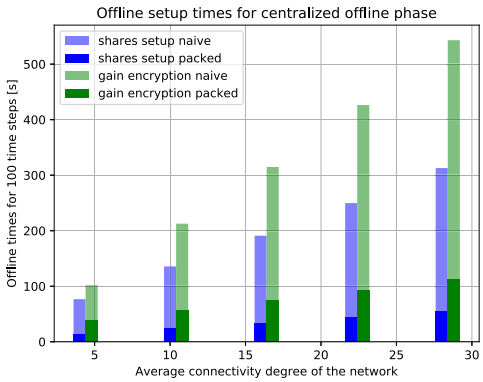


Fig. 8. Offline setup phase running times for the pWSAh* scheme in Section V in a network of 50 agents with shares generated for 100 time steps. The offline setup times for the decentralized share generation schemes in Section VI consist only of the gain encryption times.

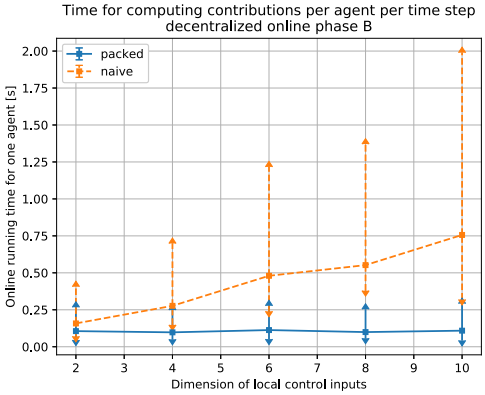


Fig. 9. Average running times for the pWSAh* scheme with the steps described in Section VI-B in a network of 25 agents.

the shares for all agents, for many time steps (see Fig. 8). Fig. 6 compares the online running times between naive and packed encryption in the case of the one-step decentralized online generation of shares and Fig. 7 for the two-step decentralized online generation of shares. These methods have roughly an eightfold increase in the online time compared to the method that makes heavy use of a trusted dealer, but the dealer has less work to do in the offline phase.

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 14:55:33 UTC from IEEE Xplore. Restrictions apply.

In the centralized offline share generation scheme, packing decreases the maximum online time between 64% and 71%. In the online decentralized cases, where the agents are also responsible for generating, encrypting, sending and decrypting the shares, packing reduces the maximum online running time between 76% and 80%. Overall, we see that in the packed version, the sampling time needs to be at most 1.1 s.

Second, the communication load is reduced when using packing: one Paillier ciphertext of 0.256 KB is sent from the neighboring agents to the aggregating agent, instead of six ciphertexts amounting to 1.536 KB, and each agent sends four batches of 16 B AES-encrypted shares to each neighbor, instead of seven batches of 16 B.

A third substantial advantage of packing is decreasing the offline time, consisting of the weights encryption and the share generation and encryption at the system operator, depicted in Fig. 8. Specifically, we see up to 80% improvement in the offline running time when using packing.

To further illustrate the efficiency of packing, we show in Fig. 9 how the performance improves with the number of control inputs, in the case of decentralized online share generation. We simulate a network of 25 agents with an average network connectivity degree of 10. Each agent has 10 local states and local control input of dimension varying between 2 and 10, which are packed in one Paillier ciphertext. The online running time remains almost the same in the packed version, despite having more control inputs, compared to the increasing online time in the naive case.

APPENDIX

A. Paillier's Additively Homomorphic Encryption Scheme

Consider the additive group of integers modulo N, $\mathbb{Z}/N\mathbb{Z}$, where $N = pq$ is a large modulus composed of two prime numbers of equal bit-length, p and q, such that $\gcd(\phi(N), N) = 1$. $\phi(N) = (p-1)(q-1)$ is the order of $(\mathbb{Z}/N\mathbb{Z})^*$. Now consider the multiplicative group of integers modulo N^2, $(\mathbb{Z}/N^2\mathbb{Z})^*$. The order of $(\mathbb{Z}/N^2\mathbb{Z})^*$ is $N\phi(N)$. An important subgroup of $(\mathbb{Z}/N^2\mathbb{Z})^*$ is

$$\begin{aligned}\Gamma_N &:= \{(1+N)^\alpha \bmod N^2 | \alpha \in \{0,\dots,N-1\}\} \\ &= \{1+\alpha N | \alpha \in \{0,\dots,N-1\}\}\end{aligned} \tag{18}$$

where the equality follows from the binomial theorem: $(1+N)^\alpha = 1+\alpha N \bmod N^2$. Computing discrete logarithms in Γ_N is easy [32], [38]: given $x, y \in \Gamma_N$, we can find β such that $y = x^\beta \bmod N^2$ by $\beta = (y-1)/(x-1) \bmod N$.

Another important subgroup in $(\mathbb{Z}/N^2\mathbb{Z})^*$ is

$$\mathfrak{G}_N := \{x^N \bmod N^2 | x \in (\mathbb{Z}/N\mathbb{Z})^*\}. \tag{19}$$

$\mathfrak{G}_N$ has order $\phi(N)$. Computing discrete logarithms in $\mathfrak{G}_N$ is as hard as computing discrete logarithms in $(\mathbb{Z}/N\mathbb{Z})^*$ [32].

We also have the modular equalities for $x \in (\mathbb{Z}/N^2\mathbb{Z})^*$:

$$x^{\phi(N)} = 1 \bmod N, \quad x^{N\phi(N)} = 1 \bmod N^2. \tag{20}$$

The Paillier scheme is defined using the previously described concepts. Specifically, the plaintext space is $\mathbb{Z}/N\mathbb{Z}$ and the ciphertext space is $(\mathbb{Z}/N^2\mathbb{Z})^*$. The public key is (g, N), where g is usually selected to be $1+N$, and the secret key $(\phi(N), (\phi(N))^{-1} \bmod N)$. The encryption is

$$\mathrm{E}(x) = g^x r^N \bmod N^2, \quad r \in (\mathbb{Z}/N\mathbb{Z})^*.$$

For a ciphertext $c \in (\mathbb{Z}/N^2\mathbb{Z})^*$, decryption uses (18) and (20)

$$\mathrm{D}(c) = (c^{\phi(N)} - 1)/N \cdot \phi(N)^{-1} \bmod N.$$

The Paillier scheme allows for homomorphic additions and multiplication by plaintexts, as follows:

$$\begin{aligned}\mathrm{D}\left(\mathrm{E}(x) \cdot \mathrm{E}(y)\right) &= \mathrm{D}\left(\mathrm{E}(x+y)\right) = x + y \bmod N \\ \mathrm{D}\left((\mathrm{E}(x))^y\right) &= \mathrm{D}\left(\mathrm{E}(xy)\right) = xy \bmod N.\end{aligned}$$

Under the decisional composite residuosity assumption (i.e., distinguishing between an element from $\mathfrak{G}_N$ and an element from $(\mathbb{Z}/N^2\mathbb{Z})^*$ is hard), the following holds:

Theorem A.1: The Paillier cryptosystem is semantically secure [38]. ◇

B. Secret Sharing

Secret sharing is a tool that distributes a secret message to a number of parties, by splitting it into random shares. Specifically, t-out-of-n secret sharing splits a secret message into n shares and distributes them to different parties; then, the secret message can be reconstructed by an authorized subset of parties, which have to combine at least t shares.

One common scheme is the additive two-out-of-two secret sharing, which splits a secret message m in a message space $\mathbb{Z}/Q\mathbb{Z}$ into two shares by: generating uniformly at random an element $s \in \mathbb{Z}/Q\mathbb{Z}$, adding it to the message and then distributing the shares s and $m + s \bmod Q$. Both shares are needed in order to recover the secret. In some cases (see discussion in Section VIII-C), we prefer to sample $s \in (0, 2^{l+\lambda})$, for a statistical security parameter λ and l the bitsize of message m.

Theorem B.2: Secret sharing is:
(a) perfectly secure when $s \in \mathbb{Z}/Q\mathbb{Z}$ [43];
(b) λ-statistically secure when $s \in (0, 2^{l+\lambda})$. ◇

Addition in $\mathbb{Z}/Q\mathbb{Z}$ involves wrap-around, ensuring that the distributions of m and $m+s$ are identical. This proves (a). The proof of (b) follows the fact that an adversary trying to distinguish between $m + s \in (0, 2^{l+\lambda} + 2^l)$ and a uniformly sampled random value $r \in (0, 2^{l+\lambda})$, has advantage $1/2 + 2^{-\lambda}$ (the statistical distance between the two sets is $2^{-\lambda}$).

C. Aggregator Obliviousness for pWSAh

We give a formal description of the privacy requirements from Section II as a cryptographic game between an adversary and a challenger, where the adversary $\mathcal{A}$ can corrupt agents and the aggregator. The weights $w_{i\in[M]}$ are constant over the time steps, so the adversary will specify two sets of constant weights: $w^{\mathcal{A},0}_{i\in[M]}$ and $w^{\mathcal{A},1}_{i\in[M]}$. The security game pWSAO (private Weighted Sum Aggregator Obliviousness) is as follows:

Setup. The challenger runs the Setup algorithm and gives the public parameters prm to the adversary.

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 14:55:33 UTC from IEEE Xplore. Restrictions apply.

Queries. The adversary can submit compromise queries and encryption queries that are answered by the challenger. In the case of compromise queries, the adversary submits an index $i \in [M]$ to the challenger and receives sk_i, which means the adversary corrupts agent i. The set of the corrupted agents is denoted by $\mathcal{C}$. In the case of encryption queries, the adversary is allowed one query per time step t and per agent $i \in [M]$. The adversary submits $(i, t, w_i^{\mathcal{A}}, x_i(t))$, where $w_i^{\mathcal{A}} = \{w_i^{\mathcal{A},0}, w_i^{\mathcal{A},1}\}$, and the challenger first runs $\mathrm{sw}_i^{\mathcal{A},0} = \mathrm{InitW}(\mathrm{prm}, i, w_i^{\mathcal{A},0})$, $\mathrm{sw}_i^{\mathcal{A},1} = \mathrm{InitW}(\mathrm{prm}, i, w_i^{\mathcal{A},1})$ and returns $\mathrm{Enc}(\mathrm{prm}, \mathrm{sw}_i^{\mathcal{A},0}, \mathrm{sk}_i, t, x_i(t))$ and $\mathrm{Enc}(\mathrm{prm}, \mathrm{sw}_i^{\mathcal{A},1}, \mathrm{sk}_i, t, x_i(t))$. The set of participants for which an encryption query was made by the adversary at time t is denoted by $\mathcal{E}(t)$.

Challenge. The adversary chooses a specific time step t^*. Let $\mathcal{U}^*$ denote the set of participants that were not compromised at the end of the game and for which no encryption query was made at time t^*, i.e., $\mathcal{U}^* = ([M] \cup \{a\}) \setminus (\mathcal{C} \cup \mathcal{E}(t^*))$. The adversary specifies a subset of participants $\mathcal{S}^* \subseteq \mathcal{U}^*$. At this time t^*, for each agent $i \in \mathcal{S}^* \setminus \{a\}$, the adversary chooses two plaintext series $x_i^0(t^*)$ and $x_i^1(t^*)$, along with $w_i^{\mathcal{A},0}$ and $w_i^{\mathcal{A},1}$, and sends them to the challenger. If $\mathcal{S}^* = \mathcal{U}^*$ and $a \notin \mathcal{S}^*$, i.e., the aggregator has been compromised, then, the values submitted by the adversary have to satisfy $\sum_{i\in\mathcal{S}^*} w_i^{\mathcal{A},0} x_i^0(t^*) = \sum_{i\in\mathcal{S}^*} w_i^{\mathcal{A},1} x_i^1(t^*)$. The challenger flips a random bit $b \in \{0, 1\}$ and computes $\mathrm{Enc}(\mathrm{prm}, \mathrm{InitW}(\mathrm{prm}, i, w_i^{\mathcal{A},b}), \mathrm{sk}_i, t, x_i^b(t^*))$, $\forall i \in \mathcal{S}^*$. The challenger then returns the ciphertexts to the adversary.

Guess. The adversary outputs a guess $b' \in \{0, 1\}$ on whether b is 0 or 1. The advantage of the adversary is defined as follows:

$$\mathbf{Adv}^{\mathrm{pWSAO}}(\mathcal{A}) := \left| \Pr[b' = b] - \frac{1}{2} \right|.$$

The adversary wins the game if it correctly guesses b.

Definition C.1: A scheme $\mathrm{pWSAh} = (\mathrm{Setup}, \mathrm{Enc}, \mathrm{InitW}, \mathrm{AggrDec})$ achieves *weighted sum aggregator obliviousness* if no probabilistic polynomial-time adversary has more than negligible advantage in winning this security game

$$\mathbf{Adv}^{\mathrm{pWSAO}}(\mathcal{A}) \leq \eta(\kappa). \qquad \diamond$$

Aggregator obliviousness implies *data privacy:* the aggregator does not learn with more than negligible probability anything about the local data of the honest agents and about their corresponding weights, other than what can be inferred from its prescribed output of the weighted sum. The game-based Definition C.1 stipulates that even when the adversary provides two sets of inputs to the protocol, it cannot distinguish which one of the inputs was used, even after being given the output.

D. Proof of Theorem 1

We treat two cases: I, the adversary does not corrupt the aggregator and II, the adversary corrupts the aggregator

$$\Pr[b' = b] = \frac{1}{2}\Pr[b' = b | i \notin \mathcal{C}] + \frac{1}{2}\Pr[b' = b | i \in \mathcal{C}].$$

We will consider the stronger case, where $\mathcal{S}^* = \mathcal{U}^*$; the weaker case, where $\mathcal{S}^* \subseteq \mathcal{U}^*$ follows.

I. $a \notin \mathcal{C}$. From the compromise queries, the adversary holds the following information $\{\kappa, \mathfrak{pk}, \{s_i(t)\}_{i\in\mathcal{C}}, \{w_i\}_{i\in\mathcal{C}}\}_{t\in[T]}$ and $\sum_{i\in\mathcal{U}} s_i(t) = -\sum_{i\in\mathcal{C}} s_i(t)$, for all $t \in [T]$. From the encryption queries at time t, the adversary knows $\{c_i(t) = \mathrm{E}(w_i^{\mathcal{A}} x_i(t)) + s_i(t))\}_{i\in\mathcal{E}(t)}$. Then, the adversary chooses $t^* \in T$ and a series of $\{x_i^0(t^*)\}_{i\in\mathcal{U}^*}$ and $\{x_i^1(t^*)\}_{i\in\mathcal{U}^*}$ and receives from the challenger $\{c_i(t^*) = \mathrm{E}(w_i^{*,b} x_i^b(t^*)) + s_i(t^*))\}_{i\in\mathcal{U}^*}$.

Because the adversary does not have the secret key of the Paillier scheme and does not have the individual secrets of the uncorrupted agents, the following holds:

$$\begin{aligned} &\Pr[\mathcal{A} \text{ breaks Paillier scheme}] \leq \eta_1(\kappa) \\ &\Pr[\mathcal{A} \text{ breaks secret sharing}] \leq \eta_2(\kappa) \\ &\Pr[b' = b | i \notin \mathcal{C}] \leq \frac{1}{2} + \eta_1(\kappa)\eta_2(\kappa) \end{aligned} \tag{21}$$

where $\eta_1(\kappa), \eta_2(\kappa)$ are negligible functions, according to Theorems A.1 and B.2.

II. $a \in \mathcal{C}$. From the compromise queries, the adversary holds $\{\kappa, \mathfrak{pk}, \{s_i(t)\}_{j\in\mathcal{C}}, \{w_i\}_{i\in\mathcal{C}}, \mathfrak{sk}\}_{t\in[T]}$, and $\sum_{i\in\mathcal{U}} s_i(t) = -\sum_{i\in\mathcal{C}} s_i(t)\ \forall t \in [T]$. From the encryption queries, and after using $\mathfrak{sk}$ to decrypt, the adversary knows $\{p_i(t) = w_i^{\mathcal{A}} x(t) + s_i(t)\}_{i\in\mathcal{E}(t)}$. Then, the adversary chooses $t^* \in T$ and a series of $\{x_i^0(t^*)\}_{i\in\mathcal{U}^*}$ and $\{x_i^1(t^*)\}_{i\in\mathcal{U}^*}$, such that $\sum_{i\in\mathcal{U}^*} w_i^{\mathcal{A},0} x_i^0(t^*) = \sum_{i\in\mathcal{U}^*} w_i^{\mathcal{A},1} x_i^1(t^*)$ and receives from the challenger $\{c_i(t^*) = \mathrm{E}(w_i^{\mathcal{A},b} x_i^b(t^*)) + s_i(t^*)\}_{i\in\mathcal{U}^*}$. The adversary uses the secret key of the Paillier scheme to decrypt the individual ciphertexts and obtains $p_i(t^*) = w_i^{\mathcal{A},b} x_i^b(t^*) + s_i(t^*) \mod N$, for $i \in \mathcal{U}^*$. Because the secret shares of zero are different for each time $t \neq t^*$, the adversary cannot infer information about the challenge query from the previous encryption queries.

Then, the probability that the adversary wins is the probability that the adversary breaks secret sharing:

$$\begin{aligned} &\Pr[\mathcal{A} \text{ breaks secret sharing}] \leq \eta_2(\kappa) \\ &\Pr[b' = b | i \in \mathcal{C}] \leq \frac{1}{2} + \eta_2(\kappa). \end{aligned} \tag{22}$$

From (21) and (22): $\mathbf{Adv}^{\mathrm{pWSAh}}(\mathcal{A}) \leq \eta_2(\kappa)$. ■

REFERENCES

[1] J.-P. Corfmat and A. S. Morse, "Decentralized control of linear multivariable systems,"*Automatica*, vol. 12, no. 5, pp. 479–495, 1976.

[2] Y. Wang, D. J. Hill, and G. Guo, "Robust decentralized control for multimachine power systems," *IEEE Trans. Circuits Syst. I: Fundam. Theory Appl.*, vol. 45, no. 3, pp. 271–279, Mar. 1998.

[3] F. Lin, M. Fardad, and M. R. Jovanović, "Augmented lagrangian approach to design of structured optimal state feedback gains," *IEEE Trans. Autom. Control*, vol. 56, no. 12, pp. 2923–2929, Dec. 2011.

[4] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, "Graph neural networks: A review of methods and applications," *Elsevier AI Open*, vol. 1, pp. 57–81, 2020.

[5] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. 5th Intl. Conf. Learn. Represent.*, 2017.

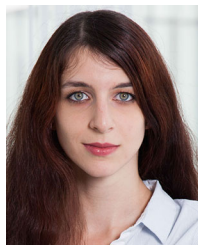
[6] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 328–339.

[7] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *J. Netw. Comput. Appl.*, vol. 116, pp. 1–8, 2018.

[8] F. Fagnani and S. Zampieri, "Average consensus with packet drop communication," *SIAM J. Control Optim.*, vol. 48, no. 1, pp. 102–133, 2009.

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 14:55:33 UTC from IEEE Xplore. Restrictions apply.

[9] S. Yang, S. Tan, and J.-X. Xu, "Consensus based approach for economic dispatch problem in a smart grid," *IEEE Trans. Power Syst.*, vol. 28, no. 4, pp. 4416–4426, Nov. 2013.
[10] B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," *Google Res. Blog*, vol. 3, 2017.
[11] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, pp. 1–19, 2019.
[12] C. J. Cleveland, R. K. Kaufmann, and D. I. Stern, "Aggregation and the role of energy in the economy," *Ecological Econ.*, vol. 32, no. 2, pp. 301–317, 2000.
[13] J. Wang, H. Zhong, C. Wu, E. Du, Q. Xia, and C. Kang, "Incentivizing distributed energy resource aggregation in energy and capacity markets: An energy sharing scheme and mechanism design," *Appl. Energy*, vol. 252, 2019, Art. no. 113471.
[14] D. Levinson and E. Chang, "A model for optimizing electronic toll collection systems," *Transp. Res. Part A, Policy Pract.*, vol. 37, no. 4, pp. 293–314, 2003.
[15] J. Balasch, A. Rial, C. Troncoso, B. Preneel, I. Verbauwhede, and C. Geuens, "Pretp: Privacy-preserving electronic toll pricing," in *Proc. 19th USENIX Conf. Secur.*, pp. 63–78, 2010.
[16] M. Schulze Darup, A. Redder, and D. E. Quevedo, "Encrypted cooperative control based on structured feedback," *IEEE Control Syst. Lett.*, vol. 3, no. 1, pp. 37–42, Jan. 2019.
[17] A. B. Alexandru, M. Schulze Darup, and G. J. Pappas, "Encrypted cooperative control revisited," in *Proc. 58th Conf. Decis. Control.*, 2019, pp. 7196–7202.
[18] A. B. Alexandru and G. J. Pappas, "Private weighted sum aggregation for distributed control systems," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 11081–11088, 2020.
[19] M. Ruan, H. Gao, and Y. Wang, "Secure and privacy-preserving consensus," *IEEE Trans. Autom. Control*, vol. 64, no. 10, pp. 4035–4049, Oct. 2019.
[20] C. N. Hadjicostis, "Privacy preserving distributed average consensus via homomorphic encryption," in *Proc. IEEE Conf. Decis. Control*, 2018, pp. 1258–1263.
[21] C. N. Hadjicostis and A. D. Dominguez-Garcia, "Privacy-preserving distributed averaging via homomorphically encrypted ratio consensus," *IEEE Trans. Autom. Control*, vol. 65, no. 9, pp. 3887–3894, Sep. 2020.
[22] E. Nozari, P. Tallapragada, and J. Cortés, "Differentially private average consensus: Obstructions, trade-offs, and optimal algorithm design," *Automatica*, vol. 81, pp. 221–231, 2017.
[23] J. He, L. Cai, P. Cheng, J. Pan, and L. Shi, "Consensus-based data-privacy preserving data aggregation," *IEEE Trans. Autom. Control*, vol. 64, no. 12, pp. 5222–5229, Dec. 2019.
[24] F. Li, B. Luo, and P. Liu, "Secure information aggregation for smart grids using homomorphic encryption," in *Proc. 1st Int. Conf. Smart Grid Commun.*, 2010, pp. 327–332.
[25] G. Ács and C. Castelluccia, "I have a DREAM!(DiffeRentially private smart metering)," in *Proc. Int. Workshop Inf. Hiding*, 2011, pp. 118–132.
[26] K. Kursawe, G. Danezis, and M. Kohlweiss, "Privacy-friendly aggregation for the smart-grid," in *Proc. Int. Symp. Privacy Enhancing Technol. Symp.*, 2011, pp. 175–191.
[27] Z. Erkin, J. R. Troncoso-Pastoriza, R. L. Lagendijk, and F. Pérez-González, "Privacy-preserving data aggregation in smart metering systems: An overview," *IEEE Signal Process. Mag.*, vol. 30, no. 2, pp. 75–86, Mar. 2013.
[28] K. Bonawitz *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1175–1191.
[29] E. Shi, H. T. H. Chan, E. Rieffel, R. Chow, and D. Song, "Privacy-preserving aggregation of time-series data," in *Proc. Netw. Distrib. Syst. Secur. Symp. Internet Soc.*, 2011.
[30] V. Rastogi and S. Nath, "Differentially private aggregation of distributed time-series with transformation and encryption," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2010, pp. 735–746.
[31] H. T. H. Chan, E. Shi, and D. Song, "Privacy-preserving stream aggregation with fault tolerance," in *Proc. Int. Conf. Financial Cryptography Data Secur.*, 2012, pp. 200–214.
[32] M. Joye and B. Libert, "A scalable scheme for privacy-preserving aggregation of time-series data," in *Proc. Int. Conf. Financial Cryptography Data Secur.*, 2013, pp. 111–125.
[33] F. Benhamouda, M. Joye, and B. Libert, "A new framework for privacy-preserving aggregation of time-series data," *ACM Trans. Inf. Syst. Secur.*, vol. 18, no. 3, 2016, Art. no. 10.
[34] D. Becker, J. Guajardo, and K.-H. Zimmermann, "Revisiting private stream aggregation: Lattice-based PSA," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018.
[35] K. Tjell and R. Wisniewski, "Privacy preserving distributed summation in a connected graph," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 3445–3450, 2020.
[36] C. Castelluccia, A. C. Chan, E. Mykletun, and G. Tsudik, "Efficient and provably secure aggregation of encrypted data in wireless sensor networks," *ACM Trans. Sensor Netw.*, vol. 5, no. 3, 2009, Art. no. 20.
[37] Q. Li, G. Cao, and T. F. La Porta, "Efficient and privacy-aware data aggregation in mobile sensing," *IEEE Trans. Dependable Secure Comput.*, vol. 11, no. 2, pp. 115–129, Mar./Apr. 2014.
[38] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. Annu. Int. Conf. Theory Appl. Cryptogr. Techn.*, 1999, pp. 223–238.
[39] D. Bickson, T. Reinman, D. Dolev, and B. Pinkas, "Peer-to-peer secure multi-party numerical computation facing malicious adversaries," *Peer-to-Peer Netw. Appl.*, vol. 3, no. 2, pp. 129–144, 2010.
[40] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: Definitions and challenges," in *Proc. Theory Cryptography Conf.*, 2011, pp. 253–273.
[41] S. Agrawal, B. Libert, and D. Stehlé, "Fully secure functional encryption for inner products, from standard assumptions," in *Proc. Annu. Int. Cryptology Conf.*, 2016, pp. 333–362.
[42] T. Ge and S. Zdonik, "Answering aggregation queries in a secure system model," in *Proc. 33rd Int. Conf. Very Large Data Bases. VLDB Endowment*, 2007, pp. 519–530.
[43] R. Cramer, I. B. Damgård, and J. B. Nielsen, *Secure Multiparty Computation*. Cambridge, U.K.: Cambridge Univ. Press, 2015.



Andreea B. Alexandru (Member, IEEE) received the B.Eng. degree in Systems Engineering from University Politehnica of Bucharest, Bucharest, Romania, in 2015 and the Ph.D. degree in electrical and systems engineering from University of Pennsylvania, Philadelphia, USA, in 2021, advised by George J. Pappas. She is currently a postdoctoral associate in the Department of Computer Science at the University of Maryland, College Park, MD, USA. Her research interests include applied cryptography, as well as the privacy and security of control systems and machine learning.

Dr. Alexandru holds three nominations for student best article awards at ICCPS, ACC and ATEE. She was the recipient of the NSF iREDEFINE Professional Development Award in 2019 and was selected as a Rising Star in EECS.



George J. Pappas (Fellow, IEEE) received the Ph.D. degree in electrical engineering and computer sciences from the University of California, Berkeley, CA, USA, in 1998.

He is currently the UPS Foundation Professor and Chair with the Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA, USA. He holds a secondary appointment with the Department of Computer and Information Sciences and the Department of Mechanical Engineering and Applied Mechanics. He is a Member of the GRASP Lab and the PRECISE Center. He was the Deputy Dean for Research with the School of Engineering and Applied Science. His research interests include control theory and, in particular, hybrid systems, embedded systems, cyber-physical systems, and hierarchical and distributed control systems, with applications to unmanned aerial vehicles, distributed robotics, green buildings, and biomolecular networks.

Dr. Pappas was the recipient of various awards, such as the Antonio Ruberti Young Researcher Prize, the George S. Axelby Award, the Hugo Schuck Best Paper Award, the George H. Heilmeier Award, the National Science Foundation PECASE Award, and numerous best student papers awards at ACC, CDC, and ICCPS.

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 14:55:33 UTC from IEEE Xplore. Restrictions apply.