Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

Private Weighted Sum Aggregation for Distributed Control Systems

Andreea B. Alexandru George J. Pappas

Department of Electrical and Systems Engineering
University of Pennsylvania, Philadelphia, PA 19104 USA
(e-mail: {aandreea,pappasg}@seas.upenn.edu*)*

Abstract: Data aggregation in distributed networks is a critical element in Internet of Things applications ranging from smart grids and robot swarms to medical monitoring over multiple devices and data centers. This paper addresses the problem of private weighted sum aggregation, i.e., how to ensure that an untrusted aggregator is able to compute *only* the weighted sum of the users' private local data, with proprietary weights. We propose a scheme that achieves the confidentiality of both the users' local data and the weights, as long as there are at least two participants that do not collude with the rest. The solution involves two layers of encryption based on the Learning With Errors problem. We discuss how to achieve efficient multi-dimensional data aggregation by using plaintext packing in the homomorphic cryptosystem used, such that the communication between the users and the aggregator is minimized.

Keywords: Data privacy, cryptography, distributed control, security, networked systems.

1. INTRODUCTION

In a plethora of applications such as learning patterns over social networks, smart grid control and autonomous vehicles coordination, there is a critical necessity of privately aggregating the distributed data. Apart from the secure communication between the participants in the computation, sophisticated methods are required to enable the confidentiality of the private data, while dealing with untrusted entities.

Recently, the area of encrypted control has gained a lot of interest from works such as Kim et al. (2016); Freris and Patrinos (2016); Farokhi et al. (2017); Schulze Darup et al. (2018b); Alexandru et al. (2018). Our current work presents further research developments in this area, focusing on encrypted distributed control. The following examples of data aggregation provide motivation for privacy-preserving methods in distributed control scenarios.

1.1 Motivating examples

Distributed control. In the context of cooperative control, the local states, system model and control gains are privacy-sensitive quantities that can reveal confidential data about the current state of an agent, as well as proprietary information about the infrastructure, e.g., in autonomous vehicle coordination and smart grids. Consider a multi-agent system with local linear dynamics:

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

where $\mathbf{x}_i \in \mathbb{R}^{n_i}$ and $\mathbf{u}_i \in \mathbb{R}^{m_i}$, for every $i \in [M]$. The agents are part of a connected and undirected communication graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with vertex set $\mathcal{V} = [M]$ and the edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. The set $\mathcal{N}_i := \{j \in \mathcal{V} | (i,j) \in \mathcal{E}\}$ represents the set of neighbors of agent i. A structured local control law can be used to stabilize the systems (1):

$$\mathbf{u}_i(t) = \mathbf{K}_{ii}\mathbf{x}_i(t) + \sum_{j \in \mathcal{N}_i} \mathbf{K}_{ij}\mathbf{x}_j(t). \qquad (2)$$

The local control law (2) results from the design of a centralized linear controller that takes into account the structural constraints of the communication graph by requiring $\mathbf{K}_{ij} = \mathbf{0}$ whenever $j \notin \mathcal{N}_i \cup \{i\}$ as in Lin et al. (2011).

The privacy of such cooperative decentralized control schemes was recently considered in Schulze Darup et al. (2018a) and Alexandru et al. (2019).

Graph neural networks. The powerful inference capabilities of neural networks have been leveraged for computing controllers Hunt et al. (1992). Recently, graph neural networks (GNN) were developed as a generalization of convolutional neural network that operate with a fixed local structure on a graph domain Zhou et al. (2018). One popular GNN architecture has the hidden state $\mathbf{h}$ at one node i at layer k described by:

$$\mathbf{h}_i^k = \sigma\left(\mathbf{W}_k^i \sum_{j \in \mathcal{N}_i \cup i} \frac{1}{\sqrt{|\mathcal{N}_i||\mathcal{N}_j|}} \mathbf{h}_j^{k-1}\right), \qquad (3)$$

where σ is a nonlinear activation function, such as sigmoid, tanh, ReLU. GNNs have also been used to compute control actions for flocking in a network, e.g. Tolstaya et al. (2019).

Oftentimes, the training is performed in a centralized manner and the model is then encrypted for the online inference step, while the hidden states at every node should remain private. The literature concerning GNNs is very recent and as far as the authors know, there are no works on the private evaluation of a GNN.

1.2 Our contributions

In this paper, we propose a private weighted sum aggregation (pWSA) scheme that uses Homomorphic Encryp-

tion and Augmented Learning with Errors. Our scheme uses the same secret keys for all the time steps of the computation and keeps the threshold of colluding agents at the cardinality of participating agents minus two. In order to make the scheme communication efficient, we use packing, which compresses a vector of messages in one plaintext, respectively one ciphertext. We show how to astutely perform the operations on the packed ciphertexts to reduce the computational and communication cost. The capabilities of the scheme proposed in this paper could also allow aggregation of nonlinear functions of private weights.

Notation. We use bold-face lower case for vectors, e.g. $\mathbf{x}$, and bold-face upper case for matrices, e.g. $\mathbf{A}$. For a positive integer n, let $[n] := \{1, 2, \ldots, n\}$. $\mathbb{Z}$ denotes the set of integers and $\mathbb{Z}_q$ denotes the ring of integers modulo q. $\mathrm{E}(pk, x)$ denotes an encrypted value x under public key pk. A function $\eta : \mathbb{Z}_{\geq 1} \to \mathbb{R}$ is called negligible if $\forall c \in \mathbb{R}_{>0}$, $\exists n_c \in \mathbb{Z}_{\geq 1}$ such that $\forall n \geq n_c$, we have $|\eta(n)| \leq n^{-c}$. $a \xleftarrow{\$} \mathcal{A}$ means that a is drawn uniformly at random from the distribution $\mathcal{A}$.

2. PROBLEM STATEMENT

Notice that the computation of the control action at one time step (2) and the evaluation of the hidden state at one layer (3) can be interpreted as an aggregation operation on weighted contributions. Specifically, consider a system with M agents and one aggregator. Each agent has some private data $\mathbf{x}_i(t) \in \mathbb{R}^n$ at time t and the aggregator wants to compute an aggregate of the data $\mathbf{x}_a(t) \in \mathbb{R}^m$:

$$\mathbf{x}_a(t+1) = \sum_{i=1}^{M} \mathbf{W}_i \mathbf{x}_i(t), \tag{4}$$

where $\mathbf{W}_i \in \mathbb{R}^{m \times n}$ are constant weights corresponding to each agent i. We will discuss the privacy solution of (4), since extending it to (2) or (3), where each agent is the aggregator of its neighbors, follows in a straightforward manner. We assume that there exists a trusted dealer, which, at the onset of the protocol, has to ensure the distribution of the weights. Figure 1 depicts the schematic representation of the weighted sum aggregation problem.

We would like a private weighted sum aggregation (pWSA) scheme to achieve the following privacy requirements:

- the aggregator can compute only the aggregate $\mathbf{x}_a(t)$ at each time period, and nothing else about the data of the agents $\mathbf{x}_i(t)$, the corresponding weights $\mathbf{W}_i$ or partial information such as $\mathbf{W}_i\mathbf{x}_i(t)$;
- without the aggregator capability, the other agents cannot learn anything about the private data and corresponding weights of the other agents;
- if the aggregator colludes with a subset of the agents (fewer than $M-1$), it inevitably learns the sum of the contributions of the remaining honest agents, but learns nothing more about their individual private data.

In addition, we require the pWSA scheme to be communication and computationally efficient. Specifically:

- only one batch of messages should be sent between the agents and the aggregator per time step.

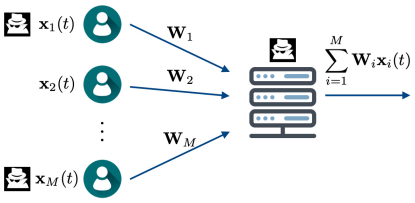

Fig. 1. The agents measure their local data $\mathbf{x}_i(t)$. The weights are supposed to be concealed from all participants. The aggregator should only learn the weighted sum of the agents' data and nothing else.

The adversarial model we consider is semi-honest, which means that an adversary wants to infer the private data of the honest participants from the messages exchanged in the protocol, without diverging from the established steps.

For simplicity, we consider the data and weights to be scalars $x_i(t), w_i \in \mathbb{R}$ in Section 3. Then, in Section 4, we show how to efficiently extend the scheme in Section 3 to vector data $\mathbf{x}_i(t) \in \mathbb{R}^{n_i}$ and matrix weights $\mathbf{W}_i \in \mathbb{R}^{m \times n_i}$.

Related work. Schulze Darup et al. (2018a) introduced a private computation and exchange of the "input portions" from the local control laws in (2), equivalent to: $\mathbf{v}_i(t) := \mathbf{W}_i\mathbf{x}_i(t)$. The aggregator would learn $\mathbf{v}_i(t)$ for all $i \in [M]$, which reveals more than simply learning the desired aggregate $\mathbf{x}_a(t)$. Details about how much private information is revealed can be found in Alexandru et al. (2019).

Alexandru et al. (2019) proposed a solution to the pWSA problem as posed in this section using additive homomorphic encryption and secret sharing. In order to guarantee privacy, the solution in Alexandru et al. (2019) required fresh secrets at every time step. These secrets were generated either offline by a trusted dealer for a large number of time steps or online, in a decentralized manner by the agents at each time step. In the decentralized solution, the cardinality of the set of colluding agents was reduced from the number of neighbors of the aggregating agent to the cardinality of the smallest intersection between the neighbors of the aggregating agent and its neighbors' neighbors.

Keeping the maximum number of colluding agents at $M-1$, reducing the involvement of a third party, as well as reducing the amount of communication are in general desirable traits of a scheme. Hence, in this work, we focus on achieving a private weighted sum aggregation scheme that does not require new correlated secrets at every time step and only involves communication between the agents and the aggregator, not also between the "non-aggregating" agents, while maintaining the natural threshold of colluding agents.

3. PRIVATE WEIGHTED SUM AGGREGATION

Private sum aggregation (pSA) allows an untrusted aggregator to compute the sum of the private data contributed by some users, without learning the individual contributions. pSA was investigated by Shi et al. (2011); Rastogi and Nath (2010); Benhamouda et al. (2016); Bonawitz et al. (2017); Becker et al. (2018). In particular, for the scheme we described in Section 2, if the agents in the network would have access to their corresponding weights $\mathbf{W}_i$, one could provide a private solution using pSA.

A private weighted sum aggregation (pWSA) scheme for weights unknown to all participants is composed of the algorithms pWSA = (Setup, Enc, AggrDec).

- Setup$(1^\kappa, M, \{w_i\}_{i\in[M]})$: given the security parameter κ, the participants and weights, generate the public parameters param, the secret information $\mathrm{sk}_i(w_i)$ for $i \in [M]$ and the aggregator secret information sk_a.
- Enc$(\mathrm{param}, \mathrm{sk}_i(w_i), x_i(t))$: Given the public parameters, the secret key of agent i and a local private plaintext, generate the corresponding ciphertext $c_i(t)$.
- AggrDec$(\mathrm{param}, \mathrm{sk}_a, \{c_i(t)\}_{i\in[M]})$: Given the public parameters, the aggregator's secret key and the ciphertexts from agents $i \in [M]$, compute the weighted sum:

$$x_a(t) = \sum_{i\in[M]} w_i x_i(t). \tag{5}$$

3.1 Formal privacy definition

We give a description of the privacy definition from Section 2 as a cryptographic game between an adversary and a challenger, where the adversary $\mathcal{A}$ can corrupt agents. The security game pWSAO (private Weighted Sum Aggregator Obliviousness) was derived from Shi et al. (2011):

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

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 $i \in [M] \cup \{a\}$ to the challenger and receives sk_i, which means the adversary corrupts agent i or the aggregator. The set of the compromised participants is denoted by $\mathcal{C}$. The cardinality of the compromised set $\mathcal{C}$ has to be strictly less than M such that the M participants cannot retrieve the key of the $M+1$'th participant, including the aggregator. 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 $(i, t, w_i^{\mathcal{A}}, x_i^{\mathcal{A}}(t))$, for constant $w_i^{\mathcal{A}}$ over the time steps, and the challenger returns $\mathrm{Enc}(\mathrm{param}, \mathrm{sk}_i(w_i^{\mathcal{A}}), x_i^{\mathcal{A}}(t))$. 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 and for which no encryption query was made at time t^*, i.e., $\mathcal{U}^* = ([M] \cup \{a\}) \setminus (\mathcal{C} \cup \mathcal{E}(t^*))$. At this time t^*, for each agent $i \in \mathcal{U}^* \setminus \{a\}$, the adversary chooses two series $x_i^0(t^*)$ and $x_i^1(t^*)$, along with $w_i^{*,0}$ and $w_i^{*,1}$, and sends them to the challenger. If $\{a\} \notin \mathcal{U}^*$, i.e., the aggregator has been compromised, then, the values submitted by the adversary have to satisfy $\sum_{i\in\mathcal{U}^*} w_i^{*,0} x_i^0(t^*) = \sum_{i\in\mathcal{U}^*} w_i^{*,1} x_i^1(t^*)$. The challenger flips a random bit $b \in \{0,1\}$ and computes $\mathrm{Enc}(\mathrm{param}, \mathrm{sk}_j(w_i^{*,b}), x_i^b(t^*))$, for all $i \in \mathcal{U}^*$, and returns the ciphertexts to the adversary.

Guess. The adversary outputs a guess b' on whether b is 0 or 1. The advantage of the adversary is defined as:

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

The adversary wins the game if it correctly guesses b.

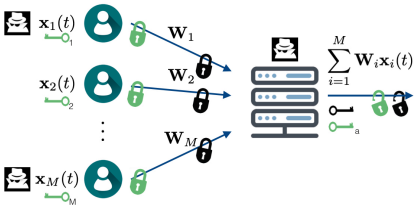

Fig. 2. Schematic representation of a scheme that achieves private weighted sum aggregation.

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

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

3.2 Preliminaries to solution

The necessary steps to achieve the private weighted sum aggregation are the following, represented also in Figure 2:

- w_i should be encrypted with an additively homomorphic encryption that the aggregator knows how to decrypt;
- the outer layer of encryption introduced in Enc should be compatible with the inner 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 inner encryption scheme.

To avoid the trust and communication issues introduced by successive symmetric keys, as used in Alexandru et al. (2019), we use a public-key additively homomorphic cryptosystem (as the inner encryption scheme) to encapsulate the message in a Learning with Errors ciphertext (the outer encryption scheme). A pSA scheme based on this idea was proposed in Becker et al. (2018) using the Augmented Learning with Errors concept introduced in El Bansarkhani et al. (2015). We present how to modify the scheme in Becker et al. (2018) such that we obtain a correct and private weighted sum aggregation scheme.

For the reader's convenience, we provided details on the *Learning with Errors* (LWE) problem in Appendix A. The LWE problem essentially amounts to distinguishing random linear equations perturbed by small amounts of noise from uniform linear equations.

Let κ denote the security parameter, $q = q(\kappa)$ a positive prime, $l = \lceil \log q \rceil$ and λ a positive integer, such that $\lambda/l \in \mathbb{Z}$. The *Augmented Learning with Errors* (A-LWE) problem El Bansarkhani et al. (2015) encodes a message in the error from an LWE term. An A-LWE term consists of $(\mathbf{A}, \mathbf{b}^\intercal)$, with $\mathbf{b}^\intercal = \mathbf{s}^\intercal\mathbf{A} + \mathbf{e}^\intercal \in \mathbb{Z}_q^\lambda$, for a public matrix $\mathbf{A} \in \mathbb{Z}_q^{\kappa\times\lambda}$, a secret key $\mathbf{s} \in \mathbb{Z}_q^\kappa$ and error term $\mathbf{e} \in \mathbb{Z}_q^\lambda$, which is sampled from a distribution related to the message we want to encode, see Appendix A.

Specifically, let μ be a message and $f(\cdot)$ a function with the property that its output is indistinguishable from random (e.g., an encryption mechanism from a semantically secure scheme). For $\mathbf{y} = f(\mu) \in \mathbb{Z}_q^{\lambda/l}$ and for a public matrix $\mathbf{G} \in \mathbb{Z}_q^{\lambda/l\times\lambda}$, the error term $\mathbf{e}$ is sampled from a special error distribution, defined by $D_{\Lambda_{\mathbf{y}}^\perp(\mathbf{G}),\sigma}$, and satisfies $\mathbf{Ge} \equiv \mathbf{y} \bmod q$. The special error distribution $D_{\Lambda_{\mathbf{y}}^\perp(\mathbf{G}),\sigma}$ is a

discrete Gaussian distribution with standard deviation σ over a lattice determined by the matrix $\mathbf{G}$ and vector $\mathbf{y}$, see Definition A.1.

Informally, given an A-LWE term $(\mathbf{A}, \mathbf{b}^\intercal)$, one cannot retrieve the secret key $\mathbf{s}$ and the message μ encoded in the error term $\mathbf{e}$, but, given $\mathbf{s}$, one can efficiently recover μ from $\mathbf{e}$, see Definition A.2.

For the inner layer of encryption, we require a semantically secure public-key additively homomorphic encryption that allows plaintext-ciphertext multiplication and the operator on the ciphertext space corresponding to addition is also addition. We will call such a scheme Packed Additively Homomorphic Encryption (PAHE) for reasons described in Section 4. For the moment, we give a bare-bones description, just to specify the compatibility with the A-LWE outer ciphertext:

- $\text{Setup}() \to \text{prm}$.
- $\text{KeyGen}(\text{prm}) \to (pk, sk)$.
- $\text{E}(pk, \mu) \to \mathbf{c}$.
- $\text{D}(sk, \mathbf{c}) \to \mu$.
- $\text{Add}(\mathbf{c}_1, \mathbf{c}_2) \leftarrow \mathbf{c} \equiv \mathbf{c}_1 + \mathbf{c}_2$.
- $\text{PMult}(p_1, \mathbf{c}_2) \leftarrow \mathbf{c} \equiv p_1 \cdot \mathbf{c}_2$.

There is an encoding step in the encryption primitive that transforms the given message into an appropriate plaintext and a decoding step in the decryption primitive that transforms the obtained plaintext into a message from the desired domain. There exist transformations between the ciphertext space, which is a ring of polynomials, and $\mathbb{Z}_q^{\lambda/l}$. So, for simplicity, we say $\mathbf{c} \in \mathbb{Z}_q^{\lambda/l}$.

3.3 Solution of pWSA problem

The idea of the scheme is as follows. The aggregator generates a pair of PAHE keys. A third party (responsible also for choosing the weights) encrypts the weights with the public key and generates a set of $M+1$ random keys that sum to zero, then distributes them accordingly to the agents and aggregator. Each agent computes the product of the encrypted weight with its local data, which is possible due to the homomorphic properties of the PAHE cryptosystem. Then, it samples the error term according to its local PAHE ciphertext, creates an A-LWE ciphertext with its local key and sends it to the aggregator. The aggregator sums all the A-LWE ciphertexts and obtains the sum of the error terms, which is an encoding of the sum of the PAHE ciphertexts. Using its PAHE secret key, the aggregator proceeds to decrypt and obtain the desired weighted sum of the data of the agents in the network.

- $\text{Setup}(1^\kappa, M, \lambda, q, \sigma, w_{1,\dots,M}, T)$: Generate the public parameters $\mathbf{A}_t \xleftarrow{\$} \mathbb{Z}_q^{\kappa \times \lambda}$, for time steps $t = 1, \dots, T$. Generate $(pk, sk) \leftarrow \text{KeyGen}$. For all agents $i \in [M]$, draw $\mathbf{s}_i \xleftarrow{\$} \mathbb{Z}_q^\kappa$ and let $\mathbf{s}_a = -\sum_{i\in[M]} \mathbf{s}_i$. For all agents $i \in [M]$, encrypt their corresponding weights by the public key $\text{E}(pk, w_i)$. Broadcast public parameters $\big(\mathbf{A}_{1,\dots,T}, q, \kappa, \sigma, \lambda, M, pk\big)$. To each agent $i \in [M]$, send their secret key $\mathbf{s}_i$ and encrypted weight $\text{E}(pk, w_i)$. Send $\big(\mathbf{s}_a, sk\big)$ to the aggregator. Each agent and the aggregator computes the vector $\mathbf{g}^\intercal = \begin{bmatrix} 1 & 2 & \dots & 2^{l-1} \end{bmatrix} \in \mathbb{Z}_q^l$. The aggregator also computes $\mathbf{G} = \mathbf{I}_{\lambda/l} \otimes \mathbf{g}^\intercal \in \mathbb{Z}_q^{\lambda/l \times \lambda}$.
- $\text{Enc}(\mathbf{A}_t, \mathbf{g}^\intercal, pk, \sigma, \mathbf{s}_i, x_i(t), \text{E}(pk, w_i))$: Each agent i computes $\mathbf{y}_i(t) = \text{PMult}(\text{E}(pk, w_i)), x_i(t)) \in \mathbb{Z}_q^{\lambda/l}$ and samples the noise term $\mathbf{e}_i(t) \leftarrow D_{\Lambda^\perp_{\mathbf{y}_i(t)}(\mathbf{G}),\sigma} \in \mathbb{Z}_q^\lambda$. Finally, it computes $\mathbf{c}_i(t) = \mathbf{s}_i^\intercal \mathbf{A}_t + \mathbf{e}_i(t)^\intercal \in \mathbb{Z}_q^\lambda$ and sends the ciphertext to the aggregator.
- $\text{AggrDec}(\mathbf{A}_t, \mathbf{G}, \mathbf{s}_a, sk, \mathbf{c}_{1,\dots,M}(t))$: The aggregator sums the ciphertexts from all the agents $\mathbf{c}(t) = \sum_{i\in[M]} \mathbf{c}_i(t)$. It then computes the aggregated error term $\mathbf{e}(t) = \mathbf{c}(t) + \mathbf{s}_a^\intercal \mathbf{A}_t$. Finally, the aggregated sum of the agents' data is computed as $\mathbf{x}_a(t) = \text{D}(sk, \mathbf{G}\mathbf{e}(t) \bmod q)$.

Correctness: The aggregator obtains the following:

$$\mathbf{x}_a(t) = \text{D}\Big(sk, \mathbf{G}\mathbf{e}(t) \bmod q\Big) \tag{6}$$
$$= \text{D}\Big(sk, \mathbf{G}\left(\mathbf{c}(t) + \mathbf{s}_a^\intercal \mathbf{A}_t\right) \bmod q\Big) \tag{7}$$
$$= \text{D}\Big(sk, \mathbf{G}\Big(\sum_{i\in[M]} \mathbf{c}_i(t) + \mathbf{s}_a^\intercal \mathbf{A}_t\Big) \bmod q\Big) \tag{8}$$
$$= \text{D}\Big(sk, \mathbf{G}\Big(\sum_{i\in[M]} \mathbf{s}_i^\intercal \mathbf{A}_t + \mathbf{e}_i(t)^\intercal + \mathbf{s}_a^\intercal \mathbf{A}_t\Big) \bmod q\Big) \tag{9}$$
$$= \text{D}\Big(sk, \mathbf{G}\Big(\sum_{i\in[M]} \mathbf{e}_i(t)^\intercal\Big) \bmod q\Big) \tag{10}$$
$$= \text{D}\Big(sk, \sum_{i\in[M]} \mathbf{G}\mathbf{e}_i(t)^\intercal \bmod q\Big) = \text{D}\Big(sk, \sum_{i\in[M]} \mathbf{y}_i\Big) \tag{11}$$
$$= \text{D}\Big(sk, \sum_{i\in[M]} \text{E}\left(pk, w_i x_i(t)\right)\Big) = \sum_{i\in[M]} w_i x_i. \tag{12}$$

The correctness of the result follows from construction: (10) follows from (9) because the keys were selected such that $\mathbf{s}_a = -\sum_{i\in[M]} \mathbf{s}_i$, (11) follows from (10) due to the linearity of $\mathbf{G}$, and the correct transition from (7) to (12) is allowed by the fact that the PAHE scheme satisfies $\text{D}(\mathbf{c}_1 + \mathbf{c}_2) = \text{D}(\mathbf{c}_1) + \text{D}(\mathbf{c}_2)$.

Theorem 1. The pWSA scheme achieves weighted sum aggregator obliviousness w.r.t. Definition 1. ◇

The proof makes use of the semantic security of the PAHE scheme and the hardness of the A-LWE problem and is provided in Appendix B.

Compared to the solution in Alexandru et al. (2019), only one set of secret shares of zero $\mathbf{s}_a = -\sum_{i\in[M]} \mathbf{s}_i$ have to be generated for all the time steps. In this case, it is reasonable to expect a trusted third party to generate and distribute them offline or even for the agents and aggregator to embark in an offline secure multi-party computation algorithm to obtain them.

For the security of the scheme over multiple time steps, we need to use different matrices $\mathbf{A}$ for each time step t, otherwise the aggregator can obtain differences of messages at two time steps. These matrices $\mathbf{A}_t$ are public and can be broadcasted or posted on a message board, where each participant has access to. For better efficiency, only smaller random seeds can be sent and stored, e.g., agree on a function that outputs a pseudorandom matrix and feed in the time steps and a smaller seed. Only one seed $\mathfrak{s}_0$ should be sent at time zero, then the agents can construct the seed for time t in a counter block cipher mode $\mathfrak{s}_t = \mathfrak{s}_0 + t$.

4. PACKED WEIGHTED AGGREGATION

In this section, we show how to extend in an efficient way the scalar weighted sum aggregation scheme to multi-dimensional data. Alexandru et al. (2019) used a naive approach where each scalar was encrypted in one ciphertext.

We require the packed additively homomorphic encryption (PAHE) scheme to be semantically secure and to satisfy the requirement from Section 3.3 of ciphertext summation. This scheme should also allow packing and single instruction multiple data (SIMD) operations. We draw inspiration from the packed additively homomorphic encryption scheme used in Juvekar et al. (2018).

PAHE can be instantiated by schemes in e.g., Brakerski and Vaikuntanathan (2011); Brakerski et al. (2014); Cheon et al. (2017). The underlying hardness problem is Ring Learning with Errors (R-LWE in Appendix A). The PAHE construction is parameterized by the following constants: the ring dimension N, the plaintext modulus p, the ciphertext modulus q and the standard deviation σ of a discrete Gaussian distribution. We can pack up to N values in one ciphertext using the Chinese Remainder Theorem. Packing can be thought of as the ciphertext having N independent data slots. Using the notation in Section 3, $N = \lambda/(2l)$.

The abstraction of the PAHE primitives are the following: PAHE.Setup, PAHE.KeyGen, PAHE.E, PAHE.D, PAHE.Eval, with the same functionalities as mentioned in the previous section. The operations that can be evaluated during PAHE.Eval are SIMDAdd, SIMDPMult and Perm, i.e. single instruction multiple data addition, element-wise multiplication by a plaintext vector and slots permutations that can achieve rotations:

- $\text{SIMDAdd}(\mathbf{c}_1, \mathbf{c}_2) \rightarrow \mathbf{c}$, such that, if $\mathbf{c}_i = \text{PAHE.E}(\boldsymbol{\mu}_i)$, $i = 1, 2$, then $\text{PAHE.D}(\mathbf{c}) = \boldsymbol{\mu}_1 + \boldsymbol{\mu}_2$.
- $\text{SIMDPMult}(\mathbf{c}, \boldsymbol{\nu}) \rightarrow \mathbf{c}'$, such that, if $\mathbf{c} = \text{PAHE.E}(\boldsymbol{\mu})$, then $\text{PAHE.D}(\mathbf{c}') = \boldsymbol{\mu} \circ \boldsymbol{\nu}$, where $\circ$ denotes element-wise multiplication.
- $\text{Perm}(\mathbf{c}, \pi) \rightarrow \mathbf{c}'$, such that, for a permutation π and $\mathbf{c} = \text{PAHE.E}(\boldsymbol{\mu})$, then $\text{PAHE.D}(\mathbf{c}') = [\boldsymbol{\mu}_{\pi(1)}, \ldots, \boldsymbol{\mu}_{\pi(n)}]$.

The encryption PAHE.E endows the ciphertexts with a fresh small noise η_0. The operations in PAHE.Eval also introduce an amount of noise in the ciphertext, which can overflow and prevent the correct decryption. Denote by η the noise level in a ciphertext $\mathbf{c}$. A ciphertext resulted from SIMDAdd has noise $\eta_1 + \eta_2$. A ciphertext resulted from SIMDPMult has noise bounded by $\eta\eta_\times$, where $\eta_\times \leq p\sqrt{N}$. A ciphertext resulted from Perm has noise $\eta + \eta_\pi$, where η_π is the noise of the permutation operation. The multiplication introduces the largest noise. In terms of computation cost, the addition is the cheapest, while the rotation is the most expensive. The parameters of the scheme are chosen such that the noise does not overflow.

4.1 Efficient homomorphic matrix-vector multiplication

In this section, we investigate the most efficient method of performing the multiplication of an encrypted matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ and plaintext $\mathbf{x} \in \mathbb{R}^n$, i.e., computing the corresponding ciphertext version of $\mathbf{y} = \mathbf{W}\mathbf{x}$. For our target applications of weighted aggregation, the goals of the computation are, in order of importance:

- minimize the size of the encrypted output that contains $\mathbf{y}$, since this ciphertext has to be encoded in the noise term and then sent over to the aggregator;
- minimize the noise growth, in order to reduce the parameter size, which in turn also minimizes both the computational cost and size of the communicated message;
- minimize the computational cost;
- minimize the input ciphertext that packs $\mathbf{W}$.

Sequential SIMDPMult operations should be avoided because of the exponential growth in the noise. For the input vector packing, we assume that $m, n < N$. For the input matrix packing, we assume that $nm < N$ or $n^2 < N$. We pad the rest of the slots up to N with zeros. If $mn > N$ (or $n^2 > N$), then the number of input ciphertexts will be $\lceil mn/N \rceil$ (respectively $\lceil n^2/N \rceil$). We show in Figure 3 the schematic representations of the five methods considered for computing a matrix-vector multiplication.

Naive method with each row packed in one ciphertext. The output of a matrix-vector multiplication is given by:

$$\mathbf{y}_j = \mathbf{W}_j \mathbf{x} = \sum_{k=1}^{n} \mathbf{W}_{jk} \mathbf{x}_k, \ j = 1, \ldots, m. \tag{13}$$

If each row of the matrix $\mathbf{W}$ is packed and encrypted in a ciphertext, and $\mathbf{x}$ is packed in a plaintext, then the naive version involves m SIMDPMult operations. For each resulting vector, we then need $n - 1$ Perm and $n - 1$ SIMDAdd operations, which creates a ciphertext j whose first slot contains $\mathbf{y}_j$. Using a tree structure to perform these operations, we can reduce their number to $\lceil \log n \rceil$ Perm and $\lceil \log n \rceil$ SIMDAdd operations. This creates m output ciphertexts. In order to obtain only one output ciphertext, we need m SIMDPMult to mask the ciphertexts by $[1|0|0|\ldots]$, then $m - 1$ Perm and $m - 1$ SIMDAdd operations. This sequence of operations introduces a lot of noise, because of the two sequential SIMDPMult operations.

Method with each column packed in one ciphertext. Denote by $\mathbf{C}_j$ the j'th column of the matrix $\mathbf{W}$, $j = 1, \ldots, n$.

$$\mathbf{y} = \sum_{j=1}^{n} \mathbf{C}_j \mathbf{x}_j. \tag{14}$$

If each $\mathbf{C}_j$ is packed and encrypted in a ciphertext, and we pack m copies of $\mathbf{x}_j$ in a plaintext, the product can be achieved by n SIMDPMult, $n - 1$ SIMDAdd operations, and no permutation, while outputting a single ciphertext.

Method with each diagonal packed in one ciphertext. In the applications mentioned in the Introduction, the matrix $\mathbf{W}$ usually satisfies $m \leq n$. We pad the matrix with zeros such that it becomes square $\bar{\mathbf{W}} \in \mathbb{R}^{n \times n}$. We can pack and encrypt every diagonal of the matrix, denoted by $\mathbf{d}_j, j = 1, \ldots, n$ as a separate ciphertext. Let $\rho(\mathbf{x}, j)$ be the rotation of $\mathbf{x}$ to the left by j elements. Then, we have:

$$\mathbf{y} = \sum_{j=1}^{n} \mathbf{d}_j \circ \rho(\mathbf{x}, j - 1). \tag{15}$$

If we rotate and pack the plaintext vector $\mathbf{x}$, the product can be achieved by n SIMDPMult, $n - 1$ SIMDAdd operations, and no permutation, while outputting a single ciphertext, the same as in the column method.

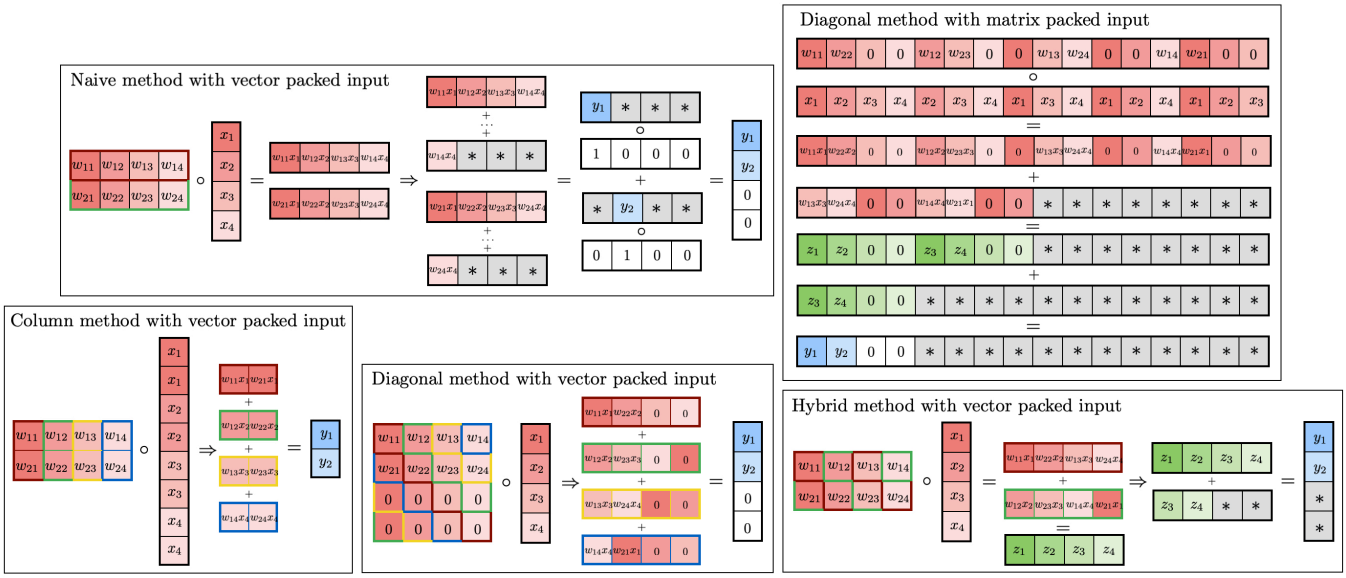

Fig. 3. Diagrams of the various matrix-vector multiplication schemes considered. The entries with the same outer coloring are packed in the same ciphertext. The inner color is selected to aid with visualizing the rotations and the corresponding element-wise slot multiplications.

Method with hybrid diagonal packing in a ciphertext. We can combine the naive and the diagonal method by packing into a ciphertext an "extended diagonal" of the rectangular matrix $\mathbf{W}$, which we denote $\bar{\mathbf{d}}_j$ for $j = 1, \ldots, m$. Then, the product can be written as:

$$\mathbf{y} = \sum_{i=1}^{\lceil n/m \rceil} \rho\Big(\sum_{j=1}^{m} \bar{\mathbf{d}}_j \circ \rho(\mathbf{x}, j-1), m(i-1) \Big). \tag{16}$$

This requires m SIMDPMult operations and $\lceil \log n/m \rceil$ Perm and $\lceil \log n/m \rceil + m - 1$ SIMDAdd.

Naive method with all matrix packed in one ciphertext. Consider that $\mathbf{W}$ is packed and encrypted in one ciphertext, row by row. Then we pack m copies of $\mathbf{x}$ in a plaintext and perform one SIMDPMult operation to get:

$$[\mathbf{W}_1 \circ \mathbf{x} | \mathbf{W}_2 \circ \mathbf{x} | \ldots | \mathbf{W}_m \circ \mathbf{x} | \ldots].$$

We only need to perform $\lceil \log n \rceil$ Perm and $\lceil \log n \rceil$ SIMDAdd operations to obtain: $[\mathbf{y}_1 | * | \ldots | * | \mathbf{y}_2 | * | \ldots | * | \mathbf{y}_m | * | \ldots]$. Notice that this yields only one output ciphertext, and no further masking is required as long as the aggregator knows which slots to retrieve when decrypting.

Method with all diagonals packed in one ciphertext. Consider that the n diagonals of $\bar{\mathbf{W}} \in \mathbb{R}^{n \times n}$ are packed and encrypted in one ciphertext. Then we can also pack the n rotated versions of $\mathbf{x}$ in a plaintext, and then perform one SIMDPMult operation to get:

$$[\mathbf{d}_1 \circ \mathbf{x} | \mathbf{d}_2 \circ \rho(\mathbf{x}, 1) | \ldots | \mathbf{d}_n \circ \rho(\mathbf{x}, n-1) | \ldots].$$

As before, we only need to perform $\lceil \log n \rceil$ Perm and $\lceil \log n \rceil$ SIMDAdd operations to obtain: $[\mathbf{y}_1 | \mathbf{y}_2 | \mathbf{y}_m | * | * | \ldots]$.

Regardless of how we pack the matrix into one ciphertext (column and hybrid packing too), the computational cost and noise are the same. The advantage of using the diagonal/column input matrix packing is that the elements of the output vector will be in the first m slots of the output ciphertext, rather than spread one every n slots.

Table 1 summarizes the number of operations, noise gain and number of input and output ciphertexts of the methods we analyzed. These methods point out a trade-off between memory and computation. The agents have to perform as many multiplications as input ciphertexts (with the exception of the naive method). At the same time, the maximum number of input ciphertexts (n) required in the diagonal/column methods does not require any permutation and has the least amount of noise.

Making a decision between the available methods should take into account the agents' capabilities and the sizes of n and m. Note that the weights are constant and transmitted only once at the protocol's initialization, hence the communication overhead for the input transmission is not decisive. For a square matrix $\mathbf{W}$, the diagonal method is the same as the hybrid one and is the default option, as is the column method. For very large n and $n >> m$, the hybrid method is preferable. For large n, m with mn comparable to N, the input matrix packing is preferable.

Remark 1. When the number of items packed in a ciphertext is less than N and rotations are performed, some slots in the output ciphertext will reveal partial sums to the decryptor. An inexpensive solution (one SIMDAdd) to this issue is to add noise to the slots that are not of interest, in order to prevent information leakage at the decryption. The diagonal and column methods with input vector packing do not require the noise treatment. ◇

4.2 Solution to pWSA multi-dimensional problem

The Setup phase unfolds as in Section 3.3: the parameters and keys are generated with respect to the PAHE cryptosystem and the weight matrices $\mathbf{W}_i$ are packed and encrypted corresponding to the chosen method from Table 1.

In the Enc phase, each agent packs its local vector $\mathbf{x}_i(t)$ in plaintext corresponding to the chosen method from Table 1 and performs that instead of PMult. Before sampling $\mathbf{e}_i(t)$ from $\mathbf{y}_i(t)$, the agents add noise as indicated in Remark 1.

Table 1. Table with costs of different methods for computing a ciphertext matrix-plaintext vector multiplication. η_0 represents the noise of the corresponding fresh ciphertext.

Method	Perm	SIMDPMult	SIMDAdd	Noise	# In ctx	# Out ctx
Naive (input vector packed)	$m\lceil \log n \rceil + m - 1$	$2m$	$m\lceil \log n \rceil + m - 1$	$m\eta_\times(n\eta_0\eta_\times + (n-1)\eta_\pi) + (m-1)\eta_\pi$	m	1
Diagonal/Column (input vector packed)	0	n	$n-1$	$n\eta_0\eta_\times$	n	1
Hybrid (input vector packed)	$\lceil \log n/m \rceil$	m	$\lceil \log n/m \rceil + m - 1$	$n\eta_0\eta_\times + \lceil \log n/m - 1 \rceil \eta_\pi$	m	1
Input matrix packed	$\lceil \log n \rceil$	1	$\lceil \log n \rceil$	$n\eta_0\eta_\times + (n-1)\eta_\pi$	1	1

In the AggrDec phase, the aggregator takes the same steps as in Section 3.3. Unlike the packed matrix-vector multiplication at the agent's side, the multiplication $\mathbf{G}\mathbf{e}(t)$ is done as is. Although the size of the matrix can be large, it is very sparse: $\mathbf{G} = \mathbf{I}_{\lambda/l} \otimes \mathbf{g}^\intercal$, where $\mathbf{g}^\intercal = [1, 2, \ldots, 2^{l-1}]$, so we only need to multiply chunks of size l. These multiplications can be efficiently obtained by bit shifting.

Correctness: The correctness of this modified scheme is immediate, given correctly selected parameters such that the noise does not overflow.

Theorem 2. The pWSA scheme for multi-dimensional problems achieves weighted sum aggregator obliviousness w.r.t. Definition 1. ◇

Proof sketch: The proof follows the same steps as the proof of Theorem 1 and also incorporates the noise added at the agents' side such that there is no information leakage from the partial sums obtained by the aggregator. □

5. CONCLUSIONS AND FUTURE WORK

We proposed a private weighted sum aggregation scheme that involves one message batch communicated at each online time step between the agents and the aggregator and minimal involvement from a third-party in a prior offline step. Moreover, we showed how to make use of input packing in order to efficiently perform homomorphic matrix-vector operations. The final efficiency of the scheme depends on the sampling scheme for encoding the messages in the error term and on the judicious choice of the parameters for the packed additive homomorphic encrypted scheme to prevent noise overflow. In our ongoing work, we are testing the proposed pWSA scheme for the decentralized control of a network of agents.

The underlying homomorphic encryption scheme can support more complex operations, at the cost of larger parameters. More specifically, we could perform computations such as:

$$\mathbf{x}_a(t) = \sum_{i \in [M]} \varphi\left(\mathbf{W}_i \mathbf{x}_i(t)\right),$$

for a nonlinear function $\varphi(\cdot)$. Our future work will explore the efficiency of such private nonlinear aggregation schemes.

REFERENCES

Alexandru, A.B., Schulze Darup, M., and Pappas, G.J. (2019). Encrypted cooperative control revisited. In *Proceedings of the IEEE Conference on Decision and Control*, 7196–7202.

Alexandru, A.B., Morari, M., and Pappas, G.J. (2018). Cloud-based MPC with encrypted data. In *Proceedings of the IEEE Conference on Decision and Control*, 5014–5019.

Becker, D., Guajardo, J., and Zimmermann, K.H. (2018). Revisiting private stream aggregation: Lattice-based PSA. In *Network & Distributed System Security Symposium*. Internet Society.

Benhamouda, F., Joye, M., and Libert, B. (2016). A new framework for privacy-preserving aggregation of time-series data. *ACM Transactions on Information and System Security*, 18(3), 10.

Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., and Seth, K. (2017). Practical secure aggregation for privacy-preserving machine learning. In *ACM SIGSAC Conference on Computer and Communications Security*, 1175–1191.

Brakerski, Z., Gentry, C., and Vaikuntanathan, V. (2014). (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory*, 6(3), 13.

Brakerski, Z. and Vaikuntanathan, V. (2011). Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 505–524. Springer.

Cheon, J.H., Kim, A., Kim, M., and Song, Y. (2017). Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, 409–437. Springer.

El Bansarkhani, R., Dagdelen, Ö., and Buchmann, J. (2015). Augmented learning with errors: The untapped potential of the error term. In *International Conference on Financial Cryptography and Data Security*, 333–352. Springer.

Farokhi, F., Shames, I., and Batterham, N. (2017). Secure and private control using semi-homomorphic encryption. *Control Engineering Practice*, 67, 13–20.

Freris, N.M. and Patrinos, P. (2016). Distributed computing over encrypted data. In *IEEE Annual Allerton Conference on Communication, Control, and Computing*, 1116–1122.

Genise, N. and Micciancio, D. (2018). Faster Gaussian sampling for trapdoor lattices with arbitrary modulus. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 174–203. Springer.

Hunt, K.J., Sbarbaro, D., Żbikowski, R., and Gawthrop, P.J. (1992). Neural networks for control systems–a survey. *Automatica*, 28(6), 1083–1112.

Juvekar, C., Vaikuntanathan, V., and Chandrakasan, A. (2018). GAZELLE: A low latency framework for secure neural network inference. In *USENIX Security Symposium*, 1651–1669.

Kim, J., Lee, C., Shim, H., Cheon, J.H., Kim, A., Kim, M., and Song, Y. (2016). Encrypting controller using fully homomorphic encryption for security of cyber-physical systems. *IFAC-PapersOnLine*, 49(22), 175–180.

Lin, F., Fardad, M., and Jovanović, M.R. (2011). Augmented Lagrangian approach to design of structured optimal state feedback gains. *IEEE Transactions on Automatic Control*, 56(12), 2923–2929.

Lyubashevsky, V., Peikert, C., and Regev, O. (2010). On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 1–23. Springer.

Rastogi, V. and Nath, S. (2010). Differentially private aggregation of distributed time-series with transformation and encryption. In *ACM SIGMOD International Conference on Management of data*, 735–746.

Schulze Darup, M., Redder, A., and Quevedo, D.E. (2018a). Encrypted cooperative control based on structured feedback. *IEEE Control Systems Letters*, 3(1), 37–42.

Schulze Darup, M., Redder, A., Shames, I., Farokhi, F., and Quevedo, D. (2018b). Towards encrypted mpc for linear constrained systems. *IEEE Control Systems Letters*, 2(2), 195–200.

Shi, E., Chan, H.T.H., Rieffel, E., Chow, R., and Song, D. (2011). Privacy-preserving aggregation of time-series data. In *Network & Distributed System Security Symposium*, 1–17. Internet Society.

Tolstaya, E., Gama, F., Paulos, J., Pappas, G., Kumar, V., and Ribeiro, A. (2019). Learning decentralized controllers for robot swarms with graph neural networks. In *Proceedings of the Conference on Robot Learning*.

Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., and Sun, M. (2018). Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*.

Appendix A. LEARNING WITH ERRORS

Let κ denote the security parameter, λ be a positive integer and $q = q(\kappa)$ a prime. Consider the ring $R = \mathbb{Z}[X]/\langle\Phi(X)\rangle$, where $\Phi(x) = x^d + 1$ is a cyclotomic polynomial with $d = 2^r$, and the quotient ring $R_q = R/qR = \mathbb{Z}_q[X]/\langle\Phi(X)\rangle$. An R-LWE term is composed of:

$$(\mathbf{a}_i, \mathbf{b}_i) \in R_q \times R_q, \text{ where } \mathbf{b}_i = \mathbf{a}_i \cdot \mathbf{s} + \mathbf{e}_i, \tag{A.1}$$

with $\mathbf{a}_i \xleftarrow{\$} R_q$ a polynomial from R_q and the secret $\mathbf{s}_i \xleftarrow{\$} R_q$. The error term $\mathbf{e}_i \in R_q$ is sampled independently according to a discretized Gaussian distribution.

The *decisional R-LWE problem* states that given polynomially many pairs $(\mathbf{a}_i, \mathbf{b}_i)$, determine whether $\mathbf{b}_i$ were constructed as in (A.1) or were randomly sampled from R_q. Informally, given these pairs, it is infeasible to recover the secret $\mathbf{s}$, see Lyubashevsky et al. (2010).

Definition A.1. [A-LWE distribution] Let κ, q, p, λ be integers and $l := \lceil \log q \rceil$. Let $\mu \in \mathbb{Z}_p$ be a plaintext and $f : \mathbb{Z}_p \to \mathbb{Z}_q^{\lambda/l}$ be a function with output indistinguishable from random. Define $\mathbf{g}^\intercal := \begin{bmatrix} 1 & 2 & \dots & 2^{l-1} \end{bmatrix} \in \mathbb{Z}_q^l$ and $\mathbf{G} := \mathbf{I}_{\lambda/l} \otimes \mathbf{g}^\intercal \in \mathbb{Z}_q^{\lambda/l \times \lambda}$. Sample $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^\kappa$ and $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{\kappa \times \lambda}$. A sample from the Augmented Learning with Errors distribution $L_{\kappa,\lambda,q}^{\text{A-LWE}}(\mu)$ over $\mathbb{Z}_q^{\kappa\times\lambda} \times \mathbb{Z}_q^\lambda$ is obtained as follows: Compute $\mathbf{y} := f(\mu) \in \mathbb{Z}_q^{\lambda/l}$; Sample $\mathbf{e} \leftarrow D_{\Lambda_{\mathbf{y}}^\perp(\mathbf{G}),\sigma} \in \mathbb{Z}_q^\lambda$; Return $(\mathbf{A}, \mathbf{b}^\intercal)$, with $\mathbf{b}^\intercal := \mathbf{s}^\intercal\mathbf{A} + \mathbf{e}^\intercal$. ◇

Let $\rho : \mathbb{R} \to (0, 1]$ with $\rho_\sigma(x) = \exp(-x^2/\sigma^2)$. The discrete Gaussian distribution over the integers $D_{\mathbb{Z},\sigma}$ samples $x \in \mathbb{Z}$ with probability $\rho_\sigma(x)/(\sum_{y\in\mathbb{Z}} \rho_\sigma(y))$. An efficient sampling algorithm of the error term $\mathbf{e}$ from the discrete Gaussian distribution $D_{\Lambda_{\mathbf{y}}^\perp(\mathbf{G}),\sigma}$ for a general q, is given by Genise and Micciancio (2018).

Definition A.2. [Decisional A-LWE problem] Let κ, q, p, λ be integers and let f be a function with output indistinguishable from random. The decisional A-LWE problem asks to distinguish in polynomial time $\text{poly}(\kappa)$ between samples $(\mathbf{A}, \mathbf{b}^\intercal) \leftarrow L_{\kappa,\lambda,q}^{\text{A-LWE}}(\mu)$ and uniform random samples $(\bar{\mathbf{A}}, \bar{\mathbf{b}}^\intercal) \xleftarrow{\$} \mathbb{Z}_q^{\kappa\times\lambda} \times \mathbb{Z}_q^\lambda$. ◇

If the distribution of $\mathbf{y}$ is computationally indistinguishable from the uniform distribution on the same domain, i.e., PAHE is semantically secure, then the decisional A-LWE problem is hard El Bansarkhani et al. (2015); Becker et al. (2018).

Appendix B. PROOF OF THEOREM 1

From the setup phase, the adversary learns the public parameters $\mathbf{A}_{t\in[T]}, q, \lambda, \kappa, \sigma, M, \mathbf{g}^\intercal, pk$ and constructed $\mathbf{G}$.

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

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

From the compromise queries, the adversary holds the following information:

I. $a \notin \mathcal{C}$. $\{\mathbf{s}_i\}_{i\in\mathcal{C}}, \{\mathrm{E}(pk, w_i)\}_{i\in\mathcal{C}}$ and $\sum_{i\in\mathcal{U}} \mathbf{s}_i = -\sum_{i\in\mathcal{C}} \mathbf{s}_i$.

II. $a \in \mathcal{C}$. $sk, \{\mathbf{s}_i\}_{i\in\mathcal{C}}, \{w_i\}_{i\in\mathcal{C}}$ and $\sum_{i\in\mathcal{U}} \mathbf{s}_i = -\sum_{i\in\mathcal{C}} \mathbf{s}_i$.

Because $\mathbf{A}_t$ is different at every time step, the adversary cannot obtain meaningful information from $\mathbf{c}_i(t_1) - \mathbf{c}_i(t_2)$.

From the encryption queries at time t, the adversary knows $\{\mathbf{c}_i(t) = \mathbf{s}_i^\intercal\mathbf{A}_t + \mathbf{e}_i(t)^\intercal\}_{i\in\mathcal{E}(t)}$, such that $\mathbf{G}\mathbf{e}_i(t) \bmod q \equiv \mathrm{E}(pk, w_i^{\mathcal{A}} x_i^{\mathcal{A}}(t))$ is satisfied for $i \in \mathcal{E}(t)$. Because PAHE is not a deterministic encryption, the adversary cannot recover information about $\mathbf{s}_i$ from computing $\mathbf{c}_i(t) - \mathbf{G}^\dagger\mathrm{E}(pk, w_i^{\mathcal{A}} x_i^{\mathcal{A}}(t)) \neq \mathbf{s}_i^\intercal\mathbf{A}_t$.

In the challenge phase, the adversary chooses $t^* \in T$ and a series of $\{x_i^0(t^*)\}_{i\in\mathcal{U}^*}$ and $\{x_i^1(t^*)\}_{i\in\mathcal{U}^*}$, $w_i^{*,0}$ and $w_i^{*,1}$.

II. $a \in \mathcal{C}$. $\sum_{i\in\mathcal{U}^*} w_i^{*,0} x_i^0(t^*) = \sum_{i\in\mathcal{U}^*} w_i^{*,1} x_i^1(t^*)$.

The challenger picks a random bit b and sends $\{\mathbf{c}_i(t^*) = \mathbf{s}_i^\intercal\mathbf{A}_{t^*} + \mathbf{e}_i^b(t^*)^\intercal\}_{i\in\mathcal{U}^*}$ to the adversary, such that $\mathbf{G}\mathbf{e}_i^b(t^*) \bmod q \equiv \mathrm{E}(pk, w_i^{*,b} x_i^b(t^*))$ holds for $i \in \mathcal{U}^*$. The adversary does not have the individual secrets of the uncorrupted agents so it cannot recover the individual error terms or keys from $\{\mathbf{c}_i(t^*)\}_{i\in\mathcal{U}^*}$ due to the hardness of A-LWE.

The adversary can sum all the ciphertexts from the encrypted queries and uncompromised set at time t^*, along with the keys from the compromised queries:

$$\begin{aligned}\sum_{i\in\mathcal{E}(t^*)} \mathbf{s}_i^\intercal\mathbf{A}_t + \mathbf{e}_i(t^*)^\intercal + \sum_{i\in\mathcal{U}(t^*)} \mathbf{s}_i^\intercal\mathbf{A}_t + \mathbf{e}_i^b(t)^\intercal + \\ + \Big(\sum_{i\in\mathcal{C}} \mathbf{s}_i\Big)^\intercal \mathbf{A}_{t^*} = \sum_{i\in\mathcal{E}(t^*)} \mathbf{e}_i(t^*)^\intercal + \sum_{i\in\mathcal{U}(t^*)} \mathbf{e}_i^b(t)^\intercal.\end{aligned} \tag{B.1}$$

Multiplying by $\mathbf{G}$, the adversary obtains:

$$\mathbf{y} = \mathrm{E}\Big(pk, \sum_{i\in\mathcal{E}(t^*)} w_i^{\mathcal{A}} x_i^{\mathcal{A}}(t^*) + \sum_{i\in\mathcal{U}(t^*)} w_i^{*,b} x_i^b(t^*)\Big). \tag{B.2}$$

II. $a \in \mathcal{C}$. The adversary uses the aggregator's key sk to decrypt (B.2) and obtains $p(t^*) = \sum_{i\in\mathcal{E}(t^*)} w_i^{\mathcal{A}} x_i^{\mathcal{A}}(t^*) + \sum_{i\in\mathcal{U}(t^*)} w_i^{*,b} x_i^b(t^*)$. Because $\sum_{i\in\mathcal{U}^*} w_i^{*,0} x_i^0(t^*) = \sum_{i\in\mathcal{U}^*} w_i^{*,1} x_i^1(t^*)$, $p(t^*)$ does not reveal any information.

Then, for both I and II, the probability that the adversary wins is the probability that it solves the A-LWE problem:

$$\Pr[\mathcal{A} \text{ solves A-LWE problem }] \leq \eta(\lambda),$$

$$\Pr[b' = b | i \notin \mathcal{C}] \leq \frac{1}{2} + \eta(\lambda), \quad \Pr[b' = b | i \in \mathcal{C}] \leq \frac{1}{2} + \eta(\lambda).$$

where $\eta(\lambda)$ is a negligible function, according to Theorem 2 in El Bansarkhani et al. (2015), Theorem 1 in Becker et al. (2018) and the semantic security of the PAHE scheme.

This results in $\mathbf{Adv}^{\text{pWSA}}(\mathcal{A}) \leq \eta(\lambda)$. □