2021 60th IEEE Conference on Decision and Control (CDC)
December 13-15, 2021. Austin, Texas

Encrypted Distributed Lasso for Sparse Data Predictive Control

Andreea B. Alexandru[†] Anastasios Tsiamis[‡] George J. Pappas[‡]

2021 60th IEEE Conference on Decision and Control (CDC) | 978-1-6654-3659-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/CDC45484.2021.9683401

***Abstract*— The least squares problem with ℓ_1-regularized regressors, called *Lasso*, is a widely used approach in optimization problems where sparsity of the regressors is desired. As motivation, we investigate a sparse data predictive control problem, run at a cloud service to control a system with unknown model, using ℓ_1-regularization to limit the behavior complexity. The collected input-output data is privacy-sensitive, hence, we design a privacy-preserving solution using homomorphically encrypted data. The main challenges are the non-smoothness of the ℓ_1-norm, which is difficult to evaluate on encrypted data, as well as the iterative nature of the Lasso problem. We use a distributed ADMM formulation that enables us to exchange substantial local computation for little communication between a few servers. We give an encrypted multi-party protocol for solving the distributed Lasso problem, by approximating the non-smooth part with a polynomial, evaluating it on encrypted data, and using a more cost effective distributed bootstrapping operation. For the example of data predictive control, we prefer a heterogeneous splitting of the data for better convergence and give an encrypted protocol for one powerful server and a few less powerful devices, added for security reasons. Finally, we provide numerical results for our proposed solutions.**

I. Introduction

Sparsity and compressed sensing have been widely used in signal processing, machine learning and control applications, especially in big-data regimes and noisy environments. In high-dimensional problems, it is likely that only a subset of features affects the observations. Pursuing sparse representations reduces the model complexity and prevents overfitting.

The celebrated Lasso algorithm (least absolute shrinkage and selection operator) accounts for both sparsity and noisy data via ℓ_1-regularization. Lasso has been used in signal reconstruction for medical imaging, wireless communication and tracking; portfolio optimization; text analysis [1], [2]. With the recent global availability and development of cloud services, outsourcing the computations is a cost effective solution when the data owner/querier lacks the computational resources or expertise to locally perform them. Given the privacy-sensitive nature of the data on which such problems are computed, and how it can be used to profile users or mount attacks on critical infrastructure, the computations should not be performed in the clear at the cloud service.

A. Contributions

To deal with the privacy issues we draw on cryptographic approaches, specifically, on homomorphic encryption, which enables polynomial computations at the cloud over the client's encrypted data. However, encrypted Lasso brings new challenges: evaluating non-smooth functions on ciphertexts, as well as continuing computations over multiple iterations and time steps, which require ciphertext refreshing.

A conventional observation is that distributing a large optimization problem to multiple servers improves the execution time by parallelizing smaller subproblems. Apart from this, we note that *distributing the computation allows a streamlined execution of encrypted iterations*. In particular, using multiple servers allows us to perform a refresh operation at a substantially reduced cost compared to performing it only at one server. This cheaper refresh operation enables us to continue the encrypted computations over multiple iterations, as well as to use a high degree polynomial to approximate the gradient of the ℓ_1-norm. Specifically, we propose:

- an efficient distributed encrypted solution to Lasso problems using ADMM, offering computational privacy of all the data, including intermediate results;
- an optimized implementation of the above protocol using an efficient Chebyshev series evaluation for polynomial approximations and reducing the number of ciphertext levels and operations.

We apply our cloud-based sparsity framework to the problem of data-based predictive control [3]–[6]. Our goal is to control an unknown system using only the privacy-sensitive (noisy) input-output data. The idea of data-driven control is to replace the state representation by a data-based representation which only uses the system trajectories, bypassing the need for system identification. In the noisy data case, inspired by [4], [6], we reformulate the data predictive control as a lasso problem. For this use-case, we propose:

- a distributed encrypted solution for ℓ_1-regularized data predictive control, using an optimized implementation.

For better convergence, we customize this solution to split the problem heterogeneously between a powerful server and a few less powerful machines, while still offering data privacy.

B. Comparison to related work

Our usage of distributed ADMM substantially differs from previous works in private distributed optimization [7]–[9]: i) we start with centralized rather than already distributed data; ii) we can split the computations heterogeneously between servers; iii) the data at each server is not in the clear, which complicates the computations; iv) the servers do not learn any of the data, including intermediate iterates and results; v) the ℓ_1-regularization term is non-smooth and has nonlinear gradient, leading to updates of the global primal variable that are incompatible with the mentioned ADMM works.

[†] Department of Computer Science, University of Maryland, College Park, MD 20742. aandreea@umd.edu

[‡] Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA 19104. {atsiamis, pappasg}@seas.upenn.edu

In [10], the authors solve a Lasso problem with distributed ADMM using secret shares and threshold additively homomorphic encryption. In contrast to their work, the data is not distributed in the clear to the computing servers, meaning we have less flexibility with respect to the local computations. Another difference is that the tools they use require them to communicate for every multiplication and comparison operations (the latter requiring a number of rounds dependent on the number of bits in the messages). In our case, the servers only send two messages per iteration and the method we employ also allows us to batch vectors and perform operations in parallel for all elements of a vector.

In [11], [12], the authors propose encrypted federated training and evaluation, using stochastic gradient descent. While we inspired our solution from their multi-party fully homomorphic encryption tool, their setup is different from ours: the data is either distributed in cleartext locally at the parties or other data providers perform the preprocessing; and the computations are different, leading to different strategies and optimizations, e.g., [11] uses a combination of distributed and centralized bootstrapping operations.

Encrypted control, surveyed in [13], offers strong privacy guarantees even when the controller is located on an untrusted platform. In the case of controlling a system with known model and linear controller parameters, [14] have shown how to perform the computations at subsequent time steps without bootstrapping or ciphertext reset. In contrast, we deal with both nonlinearities and unknown model matrices, which prevent the application of their methods.

Our previous work [15], [16] provides confidentiality for a different formulation of a data predictive control problem. The Lasso formulation in our current work does not have a closed-form solution as before, which complicates the computations on encrypted data. We also use a different architecture at the cloud, in order to *completely remove the client involvement* during the computation of the control input.

In [17], the authors propose multiple controllers in parallel that perform asynchronous local bootstrapping, to ensure that at least one has an input ready for each time step. In contrast, we prefer multiple servers to perform a distributed bootstrapping to reduce the time needed to refresh the ciphertexts.

II. Problem formulation

For a covariate matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, a vector of outcomes $\mathbf{b} \in \mathbb{R}^m$, the variable $\mathbf{x} \in \mathbb{R}^n$ and a penalty parameter $\lambda > 0$, the Lasso problem in its Lagrangian form is given by:

$$\min_{\mathbf{x}} \ \frac{1}{2}||\mathbf{A}\mathbf{x} - \mathbf{b}||_2^2 + \lambda||\mathbf{x}||_1. \tag{1}$$

For dependent covariates, there is no closed-form solution to (1). But, for instance, Lasso problems can be solved using the Alternating Direction Method of Multipliers (ADMM) [18], [2, Ch. 5]. Splitting the objective function in the ADMM way, we get:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{z}} \ & \frac{1}{2}||\mathbf{A}\mathbf{x} - \mathbf{b}||_2^2 + \lambda||\mathbf{z}||_1 \\ \text{s.t.} \ & \mathbf{x} - \mathbf{z} = \mathbf{0}. \end{aligned} \tag{2}$$

Let $S_\alpha(\mathbf{x}) = (\mathbf{x} - \alpha\mathbf{1})_+ - (-\mathbf{x} - \alpha\mathbf{1})_+$ denote the soft thresholding operator. The ADMM algorithm for (2) is:

$$\begin{aligned} \mathbf{x}^{k+1} &= \left(\mathbf{A}^\intercal\mathbf{A} + \rho\mathbf{I}\right)^{-1}\left(\mathbf{A}^\intercal\mathbf{b} + \rho(\mathbf{z}^k - \mathbf{w}^k)\right) \\ \mathbf{z}^{k+1} &= S_{\lambda/\rho}(\mathbf{x}^{k+1} + \mathbf{w}^k) \\ \mathbf{w}^{k+1} &= \mathbf{w}^k + \mathbf{x}^{k+1} - \mathbf{z}^{k+1}. \end{aligned} \tag{3}$$

While for general optimization problems, ADMM might converge slowly, for Lasso it is known to have a fast convergence of a few (tens of) iterations for a large range of the parameter $\rho > 0$ [18]. We also note that in noisy control problems, like the one we investigate in Section V, a very high precision of the optimal result is not required.

Goals and privacy requirements. A client outsources problem (1) to a cloud service that has to compute the optimal solution based on the data from the client.

We consider the cloud service to be *semi-honest*, meaning it does not deviate from the client's specifications, but can process the data it receives to extract private information for its own profit. The cloud service can be a conglomerate of K servers, possibly belonging to different organizations, offering the guarantee that not all K servers collude.

Under this adversarial model, we require *client data confidentiality*, i.e., an adversary corrupting at most $K-1$ of the servers should not be able to infer anything about the client's inputs and outputs, which consist of the values of the matrix $\mathbf{A}$ and the vector $\mathbf{b}$, any intermediate values, and solution $\mathbf{x}^*$. The penalty λ and parameter ρ can be chosen by the cloud service or the client, but are public.

III. Preliminaries

A. Homomorphic Encryption (HE)

A HE scheme is called *leveled homomorphic* if it supports the encrypted evaluation of a finite degree polynomial and *fully homomorphic* if it supports the encrypted evaluation of arbitrary polynomials. In most leveled HE schemes, operations introduce some noise; this noise accumulates and if it exceeds a threshold, decryption yields an incorrect result. We say that a fresh ciphertext is on level L and a multiplication consumes a level. A ciphertext on level 0 does not accept any more multiplications (due to overflowing noise). Leveled HE schemes can be turned into fully HE by enabling a *bootstrapping* operation, which *refreshes* the ciphertext and allows further operations while guaranteeing correct decryption.

If done locally at one server (no access to the private key), bootstrapping is a very expensive operation, consuming $\sim$10 levels; Apart from the computationally intensive refreshing procedure, all the prior operations are impacted, since ciphertexts are required to have more levels, leading to huge ciphertext sizes. Instead of performing bootstrapping locally, a server can ask the client to refresh a ciphertext on level 0. However, this implies more computation, communication and availability from the client, which is often prohibitive. An alternative that we prefer is to use multiple servers for efficient computation and refreshing. *Distributed bootstrapping trades substantial computation power for one round of communication*, as described below.

In [11], [19] a multi-party leveled HE scheme is described, where a number of servers can carry out the homomorphic computations locally and only interact for bootstrapping. The private key is additively secret shared between the servers, meaning that no proper subset of the servers can decrypt. Distributed bootstrapping requires each server to use its local secret share of the key to perform a partial decryption, mask this result and send it to the other servers. Summing up the partial decryptions results in a refreshed ciphertext with the desired number of levels that can be correctly decrypted to the original message. The distributed bootstrapping consumes $\sim$3 levels.

With this HE scheme, we can encrypt multiple scalars in a ciphertext and perform single instruction multiple data (SIMD) operations on ciphertexts: addition, element-wise multiplication and data slot rotations. This brings major computation and memory improvements. We use $+$ and $\odot$ for SIMD addition and multiplication and $\rho(\mathbf{x}, i)$ to denote the row vector $\mathbf{x}$ rotated to the left by i positions ($i < 0$ rotates to the right). We denote by $\mathrm{E_{v0}}(\mathbf{x})$ the encryption of the vector $\mathbf{x}$ followed by trailing zeros and by $\mathrm{E_{v*}}(\mathbf{x})$ the encryption of the vector $\mathbf{x}$ followed by junk elements.

A HE scheme has a *computational security parameter* κ if all known attacks against the scheme take 2^κ bit operations. In practice, at least 128 bits of security are preferred [20].

B. Polynomial approximation

Since HE can evaluate polynomials on encrypted values, we have to polynomially approximate non-polynomial functions. We choose to work with the Chebyshev series polynomial interpolation, since it is a near-minimax approximation of a continuous function on the interval $[-1, 1]$ [21].

However, polynomial approximation is not a panacea: for non-smooth functions, it gives a small error only on relatively small intervals and using high degree polynomials. We choose to use this method, rather than other more costly encrypted computation tools that can exactly evaluate non-smooth functions, knowing that we deal with noisy systems where small approximation errors are absorbed by noise.

IV. Encrypted distributed Lasso

In our setup, the client encrypts its data $\mathbf{A}, \mathbf{b}$ and splits its private key between a number of servers (each server only receives a share of the key). The servers are responsible to compute and return the solution of problem (2) to the client.

A. Challenge: Evaluating non-polynomial functions

In the steps (3) of the ADMM algorithm, the soft thresholding function is non-polynomial, yet we need to evaluate it on encrypted data when computing $\mathbf{z}^{k+1}$. We deal with this challenge by approximating it using a polynomial on a fixed interval via the Chebyshev series. The approximation error depends on the polynomial degree and the input interval.

Remark 1: The "stability" of the ADMM iterations allows $\mathbf{x}^{k+1}+\mathbf{w}^k$ to stay within a fixed interval, see Lemma 1. In practice, we choose this interval from prior simulation.

Lemma 1: Define $\mathbf{M} := \mathbf{A}^\intercal\mathbf{A}+\rho\mathbf{I}$, $\mathbf{n} := \mathbf{M}^{-1}\mathbf{A}^\intercal\mathbf{b}$ and $c := \sqrt{n}\lambda/\rho||2\rho\mathbf{M}^{-1} - \mathbf{I}||_2+||\mathbf{n}||_2$. Since $\sigma := ||\rho\mathbf{M}^{-1}||_2$ is in $(0, 1]$, we have the following bounds for the quantity $||\mathbf{x}^{k+1} + \mathbf{w}^k||_\infty$ in (3), for all $k = 1, \ldots, K_{\text{iter}}$:

$$||\mathbf{x}^{k+1} + \mathbf{w}^k||_\infty \le \sigma^k||\mathbf{n}||_2+(1-\sigma^k)/(1-\sigma)c, \quad \text{if } \sigma < 1$$
$$||\mathbf{x}^{k+1} + \mathbf{w}^k||_\infty \le ||\mathbf{n}||_2+kc, \quad \text{if } \sigma = 1.$$

Proof: Let $\mathbf{y}^k := \mathbf{x}^k + \mathbf{w}^{k-1}$ Manipulating (3), we get:

$$\begin{aligned}\mathbf{y}^{k+1} &= (\mathbf{I} - \rho\mathbf{M}^{-1})\mathbf{y}^k + (2\rho\mathbf{M}^{-1} - \mathbf{I})\mathbf{z}^k + \mathbf{n}\\ &= \rho\mathbf{M}^{-1}\mathbf{y}^k + (2\rho\mathbf{M}^{-1} - \mathbf{I})(\mathbf{z}^k - \mathbf{y}^k) + \mathbf{n}.\end{aligned}$$

The expression of the thresholding operator gives the following bound: $-\lambda/\rho \le \mathbf{z}_i^k - \mathbf{y}_i^k \le \lambda/\rho$, for $i = 1, \ldots, n$. Then, using the triangle inequality and submultiplicative property:

$$\begin{aligned}||\mathbf{y}^{k+1}||_2 &\le ||\rho\mathbf{M}^{-1}\mathbf{y}^k||_2+||(2\rho\mathbf{M}^{-1} - \mathbf{I})(\mathbf{z}^k - \mathbf{y}^k)||_2+||\mathbf{n}||_2\\ &\le ||\rho\mathbf{M}^{-1}||_2||\mathbf{y}^k||_2+\sqrt{n}\lambda/\rho||2\rho\mathbf{M}^{-1} - \mathbf{I}||_2+||\mathbf{n}||_2.\end{aligned}$$

To get the desired bounds, set $\mathbf{z}^0 = \mathbf{w}^0 = \mathbf{0}$, compress the geometric progression and use $||\mathbf{y}^{k+1}||_\infty \le ||\mathbf{y}^{k+1}||_2$. ∎

For efficiency, we implement the Paterson-Stockmayer algorithm [22], which reduces the number of homomorphic multiplications to evaluate a polynomial of degree n to $\lceil\sqrt{2n} + \log n\rceil + \mathcal{O}(1)$ (from $O(n)$ in the naive case) by recursively evaluating polynomials of smaller degree, and consumes $\lceil\log n\rceil$ levels.

B. Challenge: Evaluating iterations

The polynomial approximation could have a high degree, so to continue the subsequent iterations we need to bootstrap. We employ multiple servers to realize a cheaper bootstrapping compared to a centralized bootstrapping and a less burdensome solution than requesting client action.

We turn to the distributed version of ADMM [18], such that we use the servers both to ease the computation of the optimal solution and to ensure privacy through encrypted computations. We split the matrix $\mathbf{A}$ and vector $\mathbf{b}$ into K parts, each to be held by a server, and rewrite (2) as:

$$\begin{aligned}\min_{\mathbf{x}_1,\ldots,\mathbf{x}_K,\mathbf{z}} \ & \frac{1}{2}\sum_{i=1}^{K}||\mathbf{A}_i\mathbf{x}_i - \mathbf{b}_i||_2^2+\lambda||\mathbf{z}||_1\\ \text{s.t.} \ & \mathbf{x}_i - \mathbf{z} = \mathbf{0}, \quad i = 1, 2, \ldots, K.\end{aligned} \tag{4}$$

The ADMM algorithm for problem (4) is:

$$\begin{aligned}\mathbf{x}_i^{k+1} &= \left(\mathbf{A}_i^\intercal\mathbf{A}_i + \rho\mathbf{I}\right)^{-1}\left(\mathbf{A}_i^\intercal\mathbf{b}_i + \rho(\mathbf{z}^k - \mathbf{w}_i^k)\right)\\ \mathbf{z}^{k+1} &= \frac{1}{K}S_{\lambda/\rho}\left(\sum_{i=1}^{K}\mathbf{x}_i^{k+1} + \sum_{i=1}^{K}\mathbf{w}_i^k\right)\\ \mathbf{w}_i^{k+1} &= \mathbf{w}_i^k + \mathbf{x}_i^{k+1} - \mathbf{z}^{k+1}, \qquad i = 1, \ldots, K.\end{aligned} \tag{5}$$

Each server is given ciphertexts corresponding to $\mathbf{A}_i, \mathbf{b}_i$. We assume a preprocessing step where servers precompute convenient ciphertexts to be used in the online iterations, e.g., $1/\rho\mathbf{A}_i^\intercal\mathbf{b}_i$ and $\rho\left(\mathbf{A}_i^\intercal\mathbf{A}_i + \rho\mathbf{I}\right)^{-1}$. The servers can use the matrix inversion lemma to compute an inversion of a smaller matrix, which saves in offline computation. Online, each server locally computes the encryptions of $\mathbf{x}_i$ and $\mathbf{w}_i$, then communicates to the others the local sum $\mathbf{x}_i^{k+1}+\mathbf{w}_i^k$, so all servers are able to compute $\mathbf{z}^{k+1}$. So far, the only communication necessary is the same as in the unencrypted ADMM.

C. Challenge: Realizing the fewest bootstrapping operations

Distributed bootstrapping requires all parties to start by holding the same ciphertext that will then be refreshed. Bootstrapping the ciphertext encrypting $\mathbf{z}^{k+1}$ seems attractive, since it is global and its evaluation involves the most sequential multiplications. But $\mathbf{w}_i^{k+1}$ loses levels through $\mathbf{x}_i^{k+1}$, which is the result of a multiplication; so we would need to bootstrap both before and after computing $\mathbf{z}^{k+1}$.

Instead, we do the following trick. Each server already has to compute and send a ciphertext encrypting $\mathbf{x}_i^{k+1}+\mathbf{w}_i^k$ to the other servers in order to compute the global iterate $\mathbf{z}^{k+1}$. This means that each server can construct a packed ciphertext c^{k+1} encrypting $[(\mathbf{x}_1^{k+1}+\mathbf{w}_1^k)^\intercal \ldots (\mathbf{x}_K^{k+1}+\mathbf{w}_K^k)^\intercal]$ and distributedly bootstrap it. Afterwards, each server extracts the refreshed ciphertext containing its local value $\mathbf{x}_i^{k+1}+\mathbf{w}_i^k$, as well as a refreshed ciphertext containing $\sum_{i=1}^K \mathbf{x}_i^{k+1}+\mathbf{w}_i^k$ by repeatedly rotating and summing the refreshed ciphertext c^{k+1}. From this value, each server can locally compute its encrypted iterates $\mathbf{w}_i^{k+1}$ and $\mathbf{x}_i^{k+1}$, while bootstrapping only once per iteration rather than twice.

Apart from $\mathbf{x}_i^{k+1}+\mathbf{w}_i^k$, the servers send one more message to complete the bootstrapping, so there are two rounds of communication per iteration. In the full version [23], we analyze the number of levels this computation requires and why bootstrapping every iteration rather than once every few iterations leads to smaller parameters and ciphertexts.

D. Encrypted protocol

Protocol 1 describes the steps for privately solving the distributed Lasso problem. We use an optimized diagonal method [24] for encrypted matrix-vector multiplication. Consider a matrix $\mathbf{S}\in\mathbb{R}^{n\times n}$ and a vector $\mathbf{p}\in\mathbb{R}^n$. Denote the diagonals of $\mathbf{S}$ by $\mathbf{d}_i$, for $i=0,\ldots,n-1$. Let $n_1:=\lceil\sqrt{n/2}\rceil$ and $n_2:=n/n_1$. The corresponding result $\mathbf{q}=\mathbf{S}\mathbf{p}$ is:

$$\mathbf{q}=\sum_{i=0}^{n-1}\mathbf{d}_i\odot\rho(\mathbf{p},i) \tag{6}$$

$$=\sum_{j=0}^{n_2-1}\rho\Big(\sum_{k=0}^{n_1-1}\rho(\mathbf{d}_{j\cdot n_1+k},-j\cdot n_1)\odot\rho(\mathbf{p},k);j\cdot n_1\Big). \tag{7}$$

With (7), we need $n_1+n_2=O(\sqrt{n})$ homomorphic rotations, given $\rho(\mathbf{d}_{j\cdot n_1+k},-j\cdot n_1)$, instead of n if we use (6). In both cases we require n homomorphic multiplications.

For a rectangular matrix, we need extended diagonals but the method is the same. $\mathrm{MultDiag}(\mathbf{S},\mathbf{p})$ takes as inputs the matrix $\mathbf{S}$ as separate ciphertexts encoding diagonals rotated accordingly and the vector $\mathbf{p}$ encoded in a ciphertext with trailing zeros, and returns a ciphertext that encodes the result $\mathbf{q}$ with trailing zeros. In line 9 in Protocol 1, a masking is performed to satisfy the requirement for $\mathbf{p}$. $\mathrm{MultDiag}$ is performed locally at the servers (lines 3 and 11).

$\mathrm{ApproxSoftT}(\mathbf{p})$ implements the Chebyshev interpolation element-wise for $\mathbf{p}$, for a given set of coefficients (that specify the degree and the interval). Internally, the input is normalized to the interval $[-1,1]$ and the output is a ciphertext encoding the result with trailing zeros. $\mathrm{ApproxSoftT}$ is performed locally (line 8 in Protocol 1).

$\mathrm{DBoot}(\mathbf{p},\mathrm{sk}_i)$ is a distributed protocol, where servers start with the ciphertext of the vector $\mathbf{p}$ and each inputs their share of the private key sk_i, and servers obtain a ciphertext that contains the refreshed $\mathbf{p}$ on a predetermined level (line 6 in Protocol 1). It implies one round of communication.

Proposition 1: Protocol 1 achieves client data confidentiality with respect to semi-honest servers, assuming at least one of the servers is honest.

The proof is given in the full version of this paper [23].

PROTOCOL 1: Distributed encrypted protocol for (4).

Input: Public parameters: public key pk, number of servers K, number of iterations K_{iter}. C: secret key $\mathrm{sk}=\sum_i^K \mathrm{sk}_i$. $S_1,\ldots,S_K$: encryption of $\mathbf{M}_i=\rho(\mathbf{A}_i^\intercal\mathbf{A}_i+\rho\mathbf{I})^{-1}$, encryption of $\mathbf{m}_i=\frac{1}{\rho}(\mathbf{A}_i^\intercal\mathbf{b}_i)$, share of the secret key sk_i, the Chebyshev coefficients for evaluating $S_{\lambda/\rho}(\cdot)$ on a given interval.

Output: C: $\mathbf{x}^*$.

1: $S_{i=1,\ldots,K}$: set initial values $\mathrm{E_{v0}}(\mathbf{x}_i^0)$, $\mathrm{E_{v0}}(\mathbf{w}_i^0)$, $\mathrm{E_{v0}}(\mathbf{z}^0)$ (the value of $\mathbf{z}^k$ is previously agreed upon);
2: **for** $k=0,\ldots,K_{\mathrm{iter}}-1$ **do**
3: $\quad S_{i=1,\ldots,K}$: $\mathrm{E_{v0}}(\mathbf{x}_i^{k+1})=\mathrm{MultDiag}(\mathbf{M}_i,\mathbf{m}_i+\mathbf{z}^k-\mathbf{w}_i^k)$;
4: $\quad S_{i=1,\ldots,K}$: compute and send to other servers the rotation of the sum $\mathrm{E_{v0}}(\mathbf{v}_i):=\rho(\mathbf{x}_i^{k+1}+\mathbf{w}_i^k,-(i-1)n)$;
5: $\quad S_{i=1,\ldots,K}$: assemble $\mathrm{E_{v*}}(\mathbf{v}):=\mathrm{E_{v*}}([\mathbf{v}_1\,\mathbf{v}_2\,\ldots\,\mathbf{v}_K])$ by summing own ciphertext and all received shifted ciphertexts;
6: $\quad S_{i=1,\ldots,K}$: perform own part in the distributed bootstrapping and get $\mathrm{E_{v*}}(\mathbf{v}^b):=\mathrm{DBoot}(\mathrm{E_{v*}}(\mathbf{v}),\mathrm{sk}_i)$;
7: $\quad S_{i=1,\ldots,K}$: extract its refreshed sum of local iterates $\mathrm{E_{v*}}(\mathbf{v}_i^b)=\rho(\mathrm{E_{v*}}(\mathbf{v}^b),(i-1)n)$;
8: $\quad S_{i=1,\ldots,K}$: compute $\mathrm{E_{v*}}(\sum_{i=1}^K\mathbf{x}_i^{k+1}+\mathbf{w}_i^k)$ from $\mathrm{E_{v*}}(\mathbf{v}^b)$, then $\mathrm{E_{v0}}(\mathbf{z}^{k+1})=\mathrm{ApproxSoftT}(\frac{1}{K}\sum_{i=1}^K\mathbf{x}_i^{k+1}+\mathbf{w}_i^k,\lambda/\rho)$;
9: $\quad S_{i=1,\ldots,K}$:$\mathrm{E_{v0}}(\mathbf{w}_i^{k+1})=[\mathbf{1}_S^\intercal\,\mathbf{0}^\intercal]^\intercal\odot\mathrm{E_{v*}}(\mathbf{v}_i^b)-\mathrm{E_{v0}}(\mathbf{z}^{k+1})$;
10: **end for**
11: S_1: compute $\mathrm{E_{v0}}(\mathbf{x}_1^{K_{\mathrm{iter}}})=\mathrm{MultDiag}(\mathbf{M}_1,\mathbf{m}_1+\mathbf{z}^{K_{\mathrm{iter}}-1}-\mathbf{w}_1^{K_{\mathrm{iter}}-1})$ and send it to the client C;
12: C: decrypt and extract $\mathbf{x}^*$.

V. CASE STUDY: DATA PREDICTIVE CONTROL

Consider a linear system with unknown model parameters. A client wants to compute a reference-tracking LQR control, based only on precollected input-output data. We reformulate this control problem using the behavioral framework [3]–[5].

In (8), we construct block-Hankel matrices for the "past" and "future" input and output data, $\mathbf{u}^d\in\mathbb{R}^{mT}$ and $\mathbf{y}^d\in\mathbb{R}^{pT}$, for M samples for the past data and N samples for the future data, where $S:=T-M-N+1$ and $\mathbf{U}_p\in\mathbb{R}^{mM\times S},\in\mathbb{R}^{mN\times S}$, $\mathbf{Y}_p\in\mathbb{R}^{pM\times S}$, $\mathbf{Y}_f\in\mathbb{R}^{pN\times S}$:

$$\mathbf{H}_{M+N}(\mathbf{u}^d)=:\begin{bmatrix}\mathbf{U}_p\\ \mathbf{U}_f\end{bmatrix},\quad \mathbf{H}_{M+N}(\mathbf{y}^d)=:\begin{bmatrix}\mathbf{Y}_p\\ \mathbf{Y}_f\end{bmatrix}. \tag{8}$$

Assume we have data richness, i.e., the precollected input is *persistently exciting* [3]. This requires that the precollected input signal has length at least $(m+1)(M+N+n)-1$.

Fix a time t and let $\bar{\mathbf{u}}_t=\mathbf{u}_{t-M:t-1}$, $\bar{\mathbf{y}}_t=\mathbf{y}_{t-M:t-1}$ be the batch vector of the last M inputs and outputs, respectively. If $M\geq n$, the LQR problem can be reformulated as a data predictive control problem [4], where the state representation is replaced with precollected data. According to the behavioral framework, an input-output trajectory of a linear system is in

the image of the block-Hankel matrices for the precollected data, i.e., the constraint in (9), where $\mathbf{g}$ is a preimage of the trajectory. $\mathbf{Q}, \mathbf{R}$ are LQR costs and $\mathbf{r}$ is the desired reference. The first m elements of $\mathbf{u}^{*,t}$ are input into the system in a receding horizon fashion and $\mathbf{y}^{*,t}$ is the predicted output.

$$\begin{aligned} \min_{\mathbf{g},\mathbf{u},\mathbf{y}} \quad & \frac{1}{2}\sum_{k=t}^{N+t-1} \left(||\mathbf{y}_k - \mathbf{r}_k||^2_{\mathbf{Q}} + ||\mathbf{u}_k||^2_{\mathbf{R}}\right) \\ \text{s.t.} \quad & [\mathbf{U}_p^\intercal \ \mathbf{Y}_p^\intercal \ \mathbf{U}_f^\intercal \ \mathbf{Y}_f^\intercal]^\intercal \cdot \mathbf{g} = [\bar{\mathbf{u}}_t^\intercal \ \bar{\mathbf{y}}_t^\intercal \ \mathbf{u}^\intercal \ \mathbf{y}^\intercal]^\intercal. \end{aligned} \tag{9}$$

In practice, there is noise affecting the output measurement and precision errors induced by encryption, which might prevent an exact solution to (9). Hence, we prefer a relaxation of the equality constraints via an ℓ_2-least-squares approach with penalties λ_y and λ_u. We rewrite (9) as a minimization problem depending only on $\mathbf{g}$ by enforcing $\mathbf{u} = \mathbf{U}_f\mathbf{g}$ and $\mathbf{y} = \mathbf{Y}_f\mathbf{g}$. We also batch the objective function, using the same notation $\mathbf{Q}, \mathbf{R}, \mathbf{r}$ for the batched costs and reference.

Finally, to avoid overfitting due to noisy data, in (10) we penalize the magnitude of $\mathbf{g}$ through an ℓ_1-regularization with penalty λ_g. In the noiseless data predictive control formulation, the block-Hankel matrix of the trajectory data has an inherent low-rank structure, so choosing an ℓ_1-regularization acts like a convex relaxation of imposing a low-rank constraint–see [6, Thm. 4.6] for more details.

$$\begin{aligned} \min_{\mathbf{g}} \ & \frac{1}{2}\left(||\mathbf{Y}_f\mathbf{g} - \mathbf{r}_t||^2_{\mathbf{Q}} + ||\mathbf{U}_f\mathbf{g}||^2_{\mathbf{R}}\right) + \\ & \lambda_y||\mathbf{Y}_p\mathbf{g} - \bar{\mathbf{y}}_t||_2^2 + \lambda_u||\mathbf{U}_p\mathbf{g} - \bar{\mathbf{u}}_t||_2^2 + \lambda_g||\mathbf{g}||_1. \end{aligned} \tag{10}$$

Notice that (10) is indeed a Lasso problem:

$$\min_{\mathbf{g}} \ \frac{1}{2}||\mathbf{H}\mathbf{g} - \mathbf{J}\mathbf{f}_t||_2^2 + \lambda_g||\mathbf{g}||_1, \tag{11}$$

where we have $\mathbf{J} := \text{blkdiag}\left(2\lambda_y\mathbf{I},\ \mathbf{Q},\ 2\lambda_u\mathbf{I},\ \mathbf{R}\right)^{1/2}, \mathbf{f}_t := [\bar{\mathbf{y}}_t^\intercal \ \mathbf{r}_t^\intercal \ \bar{\mathbf{u}}_t^\intercal \ \mathbf{0}^\intercal]^\intercal, \mathbf{H} := \mathbf{J}[\mathbf{Y}_p^\intercal \ \mathbf{Y}_f^\intercal \ \mathbf{U}_p^\intercal \ \mathbf{U}_f^\intercal]^\intercal$.

Our goal now is to provide a solution that outsources to a cloud service the computation of the optimal solution $\mathbf{g}^{*,t}$ of (11) and of $\mathbf{u}^{*,t} = \mathbf{U}_f\mathbf{g}^{*,t}$, while ensuring client data confidentiality for all time steps, as described in Section II.

VI. Encrypted data predictive control

Following the discussion in Section IV, we write problem (11) as a distributed problem with split variables:

$$\begin{aligned} \min_{\mathbf{g}_1,\ldots,\mathbf{g}_K,\mathbf{z}} \ & \frac{1}{2}\sum_{i=1}^{K}||\mathbf{H}_i\mathbf{g}_i - (\mathbf{J}\mathbf{f}_t)_i||_2^2 + \lambda_g||\mathbf{z}||_1 \\ \text{s.t.} \ & \mathbf{g}_i - \mathbf{z} = \mathbf{0}, \quad i = 1, 2, \ldots, K. \end{aligned} \tag{12}$$

Problem (12) needs to be solved for each time step t and $\mathbf{f}_t$ needs to be updated with the latest input and output values.

In this particular context, when we perform a homogenous split of the data (splitting the component matrices $\mathbf{U}_p, \mathbf{U}_f, \mathbf{Y}_p, \mathbf{Y}_f$ of $\mathbf{H}$ equally between the K servers), the distributed solution converges very slowly to the global optimal solution. The reason behind this is that each server solves a local optimization problem for the same system that generated the values, but being given fewer samples than necessary to characterize the behaviour of the system, i.e., losing persistency of excitation.

To avoid this issue, we prefer to unequally split the problem. Specifically, we designate Server 1 to have most of the rows and the rest of the servers to hold fewer. Because the local solution of Server 1 is close to the central solution, the empirical convergence to the optimal solution is much faster.

Let the matrix $\mathbf{H}_1$ denote the first $(m+p)M+pN$ rows of matrix $\mathbf{H} \in \mathbb{R}^{(m+p)(N+M)\times S}$. We split the remaining rows of $\mathbf{H}$ into blocks of $mN/(K-1)$ rows, denoted $\mathbf{H}_i$ for $i = 2, \ldots, K$. We similarly split $\mathbf{H}^\intercal\mathbf{J} \in \mathbb{R}^{S\times(m+p)(N+M)}$ into $\bar{\mathbf{H}}_1$ and $\bar{\mathbf{H}}_2 \ldots, \bar{\mathbf{H}}_K$ and $\mathbf{f}_t \in \mathbb{R}^m$ into $\mathbf{f}_{1,t}$ and $\mathbf{f}_{2,t}, \ldots, \mathbf{f}_{K,t}$. We prefer to use more of *less powerful machines* in order to increase the security threshold (by splitting the secret key into more values) and reduce the cost of operating the cloud service. To this end, we shift some of the computations from the less powerful servers to the more powerful Server 1 and remove online communication between the client and the less powerful servers. We describe below the steps of the protocol for encrypted non-homogeneous ADMM for problem (10) and provide the pseudocode in the full version [23].

First, we choose the data split so $\mathbf{f}_{i,t} = 0$, for $i > 1$. Second, Servers $2, \ldots, K$ have an easier offline computation, since they have to invert substantially smaller matrices than Server 1 (via the matrix inversion lemma). The bootstrapping step is done the same as in Protocol 1, after all parties broadcast their local sums. However, we let only the more powerful Server 1 perform the summation $\sum_{i=1}^{K} \mathbf{g}_i^{k+1} + \mathbf{w}_i^k$ and the evaluation of the soft thresholding approximation, and then send the result $\mathbf{z}^{k+1}$ to the others. Then, all servers continue with the computation of $\mathbf{w}_i^{k+1}$ and finish the iteration.

Because the elements of $\mathbf{f}_t$ corresponding to Servers $2, \ldots, K$ are 0, there is no need for them to update with the latest values of $\mathbf{u}_t$ and $\mathbf{y}_t$. This way, only Server 1 needs to have a connection with the client. The ciphertexts communicated to the client are on level 0 (the predicted input $\mathbf{u}^{*,t}$).

VII. Numerical results

We consider a data-driven temperature control of a 4x4 stable system representing a building with four rooms, with sampling time 300 seconds, and $M = 4$, $N = 8$, $T = 84$. We add zero mean Gaussian process and measurement noise with covariance $0.01\mathbf{I}$. The cost and regularization parameters are $\mathbf{Q} = 300\mathbf{I}, \mathbf{R} = \mathbf{I}, \lambda_g = 300, \lambda_y = \lambda_u = 3000$. The data was distributed among 3 servers: Server 1 holds 64 rows and Servers 2 and 3 hold 16 rows each. Convergence for the Lasso problem associated to one time step occurred after 20 ADMM iterations, for $\rho = 1200$. Figure 1 reflects the tracking performance of the data predictive control problem.

We evaluated the protocol from Section VI on Ubuntu 18.04 on a commodity laptop with 8 GB of RAM and Intel Core i7, implemented using the PALISADE library [25], using 8 threads. We set a security level of 128 bits using a ciphertext modulus of 436 bits and a ring dimension of 2^{14}. We obtained 6 decimal places precision for the results.

The average time for an iteration is 1.98 seconds. The time for Server 1 to assemble the vector $\mathbf{f}_{1,t}$ from the client and to compute the prediction is 0.61 seconds. The client needs 0.07 seconds to decrypt the control input and to encrypt the new

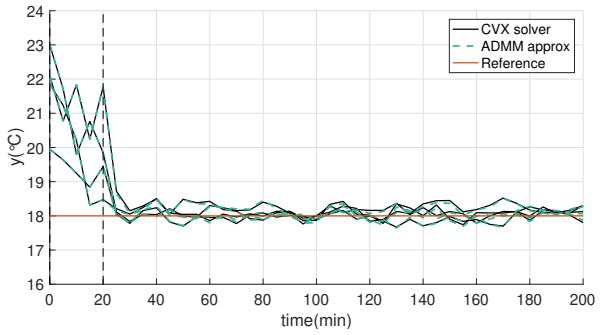

Fig. 1. Comparison between the tracking performance (temperature in 4 rooms) of the data predictive controller solving (11) exactly via the CVX solver, and via distributed ADMM with 3 servers and approximating the soft thresholding function with a degree-11 polynomial. The vertical dashed line marks the first M time steps, corresponding to the initial offline data.

measurement and the input. The total computation time per solving the optimization problem is 38.44 seconds, not taking communication into account. The setup takes 4.5 seconds, and is performed once for all subsequent iterations. Overall, the maximum amount of memory Server 1 needs to have is 1.22 GB, while Server 2 and 3 need 0.52 GB.

To simulate less powerful devices, we run Servers 2 and 3 on 2 threads instead of 8. The total time necessary for the 20 iterations increases to 49.96 seconds. The majority of the difference comes from the final operation of bootstrapping. Because the servers only need to synchronize in order to bootstrap, Server 1 can either wait for the other servers to finish the computation or can perform two local updates of $\mathbf{g}_1$. Empirically, performing more local iterations at Server 1 helps convergence speed. Nevertheless, the computation of the control input is ready in one sixth of the sampling time.

Figure 2 depicts how the time for one ADMM iteration varies with the dimension of the problem, i.e., number of columns of $\mathbf{A}$ in (2). The blue bar shows the time for lines 3, 4 in Protocol 1, effectively consisting of the matrix-vector product. The yellow bar shows the time for lines 5, 6, representing the preparation for bootstrapping and the bootstrapping itself. The red bar represents lines 7–9, consisting mostly of the polynomial evaluation. We stress that through packing, we made the bootstrapping and polynomial evaluation independent from the dimension of the problem. However, at large dimensions the encrypted matrix multiplication takes most of the computational and memory effort, and other methods that decrease storage and operations at the cost of more levels might be preferable.

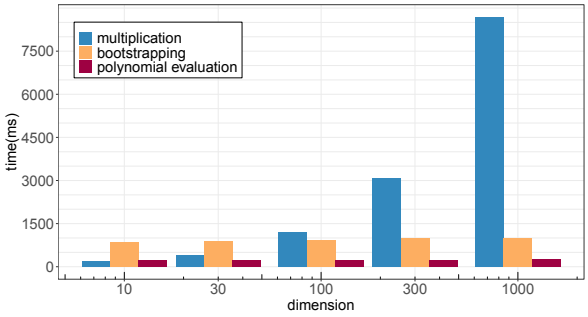

Fig. 2. Timing at one server for one iteration of solving Lasso problems of various dimensions via encrypted distributed ADMM with 3 servers (Protocol 1). The legend shows different steps in an iteration.

References

[1] Z. Zhang, Y. Xu, J. Yang, X. Li, and D. Zhang, "A survey of sparse representation: algorithms and applications," *IEEE access*, vol. 3, pp. 490–530, 2015.

[2] T. Hastie, R. Tibshirani, and M. Wainwright, *Statistical learning with sparsity: the lasso and generalizations*. CRC press, 2015.

[3] J. C. Willems, P. Rapisarda, I. Markovsky, and B. L. De Moor, "A note on persistency of excitation," *Systems & Control Letters*, vol. 54, no. 4, pp. 325–329, 2005.

[4] J. Coulson, J. Lygeros, and F. Dörfler, "Data-enabled predictive control: In the shallows of the DeePC," in *18th European Control Conf.* IEEE, 2019, pp. 307–312.

[5] J. Berberich, J. Köhler, M. A. Muller, and F. Allgower, "Data-driven model predictive control with stability and robustness guarantees," *IEEE Trans. on Automatic Control*, 2020.

[6] F. Dörfler, J. Coulson, and I. Markovsky, "Bridging direct & indirect data-driven control formulations via regularizations and relaxations," *arXiv preprint arXiv:2101.01273*, 2021.

[7] C. Zhang, M. Ahmad, and Y. Wang, "ADMM based privacy-preserving decentralized optimization," *IEEE Trans. on Information Forensics and Security*, vol. 14, no. 3, pp. 565–580, 2018.

[8] Y. Ye, H. Chen, M. Xiao *et al.*, "Privacy-preserving incremental ADMM for decentralized consensus optimization," *IEEE Trans. on Signal Processing*, vol. 68, pp. 5842–5854, 2020.

[9] Y. Lu and M. Zhu, "Privacy preserving distributed optimization using homomorphic encryption," *Automatica*, vol. 96, pp. 314–325, 2018.

[10] W. Zheng, R. Popa, J. E. Gonzalez, and I. Stoica, "Helen: Maliciously secure coopetitive learning for linear models," in *IEEE Symposium on Security and Privacy*, vol. 1, 2019.

[11] D. Froelicher, J. R. Troncoso-Pastoriza, A. Pyrgelis *et al.*, "Scalable privacy-preserving distributed learning," *Proc. on Privacy Enhancing Technologies*, vol. 2021, no. 2, pp. 323–347, 2021.

[12] S. Sav, A. Pyrgelis, J. R. Troncoso-Pastoriza *et al.*, "POSEIDON: Privacy-preserving federated neural network learning," *NDSS Symposium*, 2021.

[13] M. S. Darup, A. B. Alexandru, D. E. Quevedo, and G. J. Pappas, "Encrypted control for networked systems: An illustrative introduction and current challenges," *IEEE Control Systems Magazine*, vol. 41, no. 3, pp. 58–78, 2021.

[14] J. Kim, H. Shim, and K. Han, "Dynamic controller that operates over homomorphically encrypted data for infinite time horizon," *arXiv preprint arXiv:1912.07362*, 2019.

[15] A. B. Alexandru, A. Tsiamis, and G. J. Pappas, "Towards private data-driven control," in *59th Conf. on Decision and Control*. IEEE, 2020, pp. 5449–5456.

[16] ——, "Data-driven control on encrypted data," *arXiv preprint arXiv:2008.12671*, 2020.

[17] J. Kim, C. Lee, H. Shim *et al.*, "Encrypting controller using fully homomorphic encryption for security of cyber-physical systems," *IFAC-PapersOnLine*, vol. 49, no. 22, pp. 175–180, 2016.

[18] S. Boyd, N. Parikh, and E. Chu, *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

[19] C. Mouchet, J. Troncoso-Pastoriza, J.-P. Bossuat, and J.-P. Hubaux, "Multiparty homomorphic encryption from ring-learning-with-errors," *Proc. on Privacy Enhancing Technologies*, vol. 4, pp. 291–311, 2021.

[20] M. R. Albrecht, M. Chase, H. Chen *et al.*, "Homomorphic encryption security standard," HomomorphicEncryption.org, Tech. Rep., 2018.

[21] J. C. Mason and D. C. Handscomb, *Chebyshev polynomials*. CRC press, 2002.

[22] M. S. Paterson and L. J. Stockmeyer, "On the number of nonscalar multiplications necessary to evaluate polynomials," *SIAM Journal on Computing*, vol. 2, no. 1, pp. 60–66, 1973.

[23] A. B. Alexandru, A. Tsiamis, and G. J. Pappas, "Encrypted distributed Lasso for sparse data predictive control," *arXiv preprint arXiv:2104.11632*, 2021.

[24] S. Halevi and V. Shoup, "Algorithms in HElib," in *Annual Cryptology Conf.* Springer, 2014, pp. 554–571.

[25] "PALISADE Lattice Cryptography Library (release 1.10.4)," https://palisade-crypto.org/, Sep. 2020.