Compact Storage for Homomorphic Encryption

Adi Akavia[1⋆], Neta Oren[1], Boaz Sapir[2], and Margarita Vald[2]

[1] University of Haifa, Israel adi.akavia,neta5128@gmail.com
[2] Intuit Israel Inc., Israel Boaz_Sapir@intuit.com,margarita.vald@cs.tau.ac.il

Abstract. Homomorphic encryption (HE) is a promising technology for protecting data in use, with considerable recent years progress towards attaining practical runtime performance. However the high storage overhead associated with HE remains an obstacle preventing its large scale adoption. In this work we propose a new storage solution in the two-server model resolving the high storage overhead associated with HE, while preserving data confidentiality. Our solution attains the following desired properties:

1. *Compact storage* with zero overhead over storing AES ciphertexts, and $10\times$ to $10^4\times$ better than storing CKKS ciphertexts.
2. *Fast runtime performance* for storage and retrieval, only twice the time of directly storing and retrieving HE ciphertexts.
3. *Dynamic control during retrieval* of the HE parameters and the data items to be packed in each HE ciphertext.
4. *Plug-and-play compatibility* with any homomorphic computation.

We implemented our solution into a proof-of-concept system running on AWS EC2 instances with AWS S3 storage, empirically demonstrating its appealing performance. As a central tool we introduce the first perfect secret sharing scheme with fast homomorphic reconstruction over the reals; this may be of independent interest.

Keywords: homomorphic encryption, storage, secret sharing

1 Introduction

The privacy and safety of individuals and organizations are threatened by the ubiquity of data collected by products and services as part of the so-called AI revolution. Data collecting companies are driven to address this threat to avoid reputational damage in case of a data breach and to comply with privacy regulations such as GDPR [2] and CCPA [16]. A promising approach for enhancing data protection is to use secure computation (MPC) [63, 37] and homomorphic encrypting (HE) [58, 33] that support data protection at all times, not only at rest and at transit, but

⋆ The work was supported in part by the Israel Science Foundation grant 3380/19, and the Israel National Cyber Directorate via Haifa, BIU and Tel-Aviv Cyber Centers.

also for data in use. The HE approach is particularly appealing due to its compatibility with existing dataflow, supporting computations over encrypted data by stand-alone servers (computing server) requiring no interaction, and the complexity of all other parties, if any exists, is independent of the complexity of the evaluated function (unlike in MPC). Indeed there has been much progress towards attaining practical runtime for HE-based solutions, e.g., for privacy preserving machine learning (PPML) as in [36, 51, 59, 41, 43, 47, 23, 46, 35, 13, 12, 5, 4, 57]. These solutions often operate over massive datasets (e.g. in training machine learning models) that must be stored in HE-encrypted form. More generally, as HE-based solutions are becoming faster and faster (due to improvements in schemes, algorithms and hardware), more data is to be HE-encrypted and stored.

The high storage overhead associated with HE encryption however is a significant obstacle preventing large-scale adoption. Concretely, storing HE ciphertexts of state-of-the-art HE schemes and implementations (e.g. CKKS scheme [22] in Micorosft SEAL library [60]) incurs $10\times$ to $10^4\times$ blowup in storage size compared to storing the data in cleartext or encrypted by standard schemes such as AES. The blowup rate depends on the number of data items packed in each ciphertext, with $10\times$ blowup at maximal packing capacity which is not always applicable; for example, if data must be encrypted in a streaming fashion, or arrives from distinct sources with few data items each, or the computation is not suitable for entry-wise vector operations as supported on packed data, then packing cannot be fully utilized. Furthermore, although it is a well-known fact that storage costs are rapidly declining, for some of the use cases being proposed for HE, such as secure computation within the enterprise, the relevant storage is entire enterprise-level datalakes, at petabyte scale or more. Even the low-end of $10\times$ blowup in storage cost, at such scale, is clearly unacceptable from both a financial and operational perspective.

The question initiating this research is:

Can the high storage overhead of HE be eliminated?

1.1 Our Contribution

In this work affirmatively resolve the above question by proposing: (1) a new storage solution in the two-server model eliminating the high storage overhead associated with HE; (2) a proof-of-concept system demonstrating the appealing storage-size and runtime performance attained by our

solution. (3) As a central tool we introduce the first perfect secret sharing scheme supporting fast homomorphic reconstruction over the reals.

Contribution 1: Compact storage with oblivious HE-retrieval. We present a new storage and retrieval solution that supports retrieving data in HE-encrypted form without revealing any information on the underlying data (oblivious HE-retrieval), while maintaining storage size as small as when storing data encrypted by standard encryption schemes such as AES (compact storage) and fast storage and retrieval runtime, only twice the time of directly storing and retrieving HE ciphertexts. Our solution is generic and can be instantiated with any public key HE scheme supporting additive homomorphism, where the plaintext addition may be over a finite ring as in Paillier [56] and BGV [15] or over the reals as in CKKS [22].

Our solution consists of two protocols, *store* and *retrieve*. *Store* is a non-interactive protocol allowing multiple data producers to upload data to a common storage (e.g. a datalake on AWS S3), where data can be uploaded in blocks or one-by-one, possibly over time and interleaved with retrieval queries. *Retrieve* is a single-round interactive protocol between the computing server that obtains as output the HE encrypted data and an auxiliary service that has no output. Security holds in the two-server model, i.e., when the computing server and auxiliary service are non-colluding. In this model we guarantee correctness against semi-honest adversaries and privacy against malicious ones.

Integration with executing any homomorphic computation on the obliviously retrieved data is supported by our solution, simply by plugging in the retrieved HE ciphertexts as input to the homomorphic computation ("retrieve-then-evaluate").

To support integration with a wide range of computations, our solution supports *dynamic control* at the time of data retrieval for: (i) *Data:* fine-grained dynamic choice of the data items to be retrieved is supported, allowing retrieval of any subset of the data items. (ii) *HE scheme and parameters:* dynamic choice of the HE scheme with which the data is encrypted, its public key and context parameters such as the degree of the cyclotomic polynomial, is likewise supported. (iii) *Batching:* flexibly choosing which data items are packed together in each ciphertext is supported. The flexible dynamic control allows tailoring each oblivious retrieval to best optimize the computation at hand; for example, dynamically choosing a high degree cyclotomic polynomial in CKKS to maintain correctness if integrating with a deep homomorphic computation, and low

degree otherwise to speedup performance. Moreover, the support for on-the-fly key generation enables using ephemeral keys for enhanced safety.

This offers a viable industry compatible solution for computing on HE-encrypted data with zero data exposure during the pipeline while eliminating the high storage overhead associated with storing HE ciphertexts.

Discussion on the credibility and usability of the two-server model. In the integrated protocol, the two non-colluding servers are required only during the oblivious retrieval phase, whereas the homomorphic evaluation phase is executed by a single server solely carrying the entire burden of the computation. This is in contrast to prior work in the two-server model, including those utilizing HE [54, 35, 5, 7, 42], that required two non-colluding servers throughout the entire computation. Namely, in our work, the non-collusion assumption is limited only to the brief oblivious retrieval phase and not present throughout the bulk of the computation. Moreover, the auxiliary service in our oblivious retrieval has a predefined lightweight logic. This is in sync with existing enterprise best practices, where services with a simple predefined logic (e.g., a key management or our auxiliary service) can be effectively safeguarded, making the non-collusion assumption credible.

Contribution 2: Proof-of-concept system implemented on AWS. We present a proof-of-concept system implementing our compact storage with oblivious HE-retrieval while using CKKS [22] as the HE scheme and our secret sharing scheme as a central tool.

The system is implemented in AWS cloud computing environment with a separate EC2 instance (M5.2xlarge) for each participating entity (data producer, computing server and auxiliary service), HTTP communication between the latter two, and S3 bucket for storage. We enforced access control to S3 by applying AES encryption on uploaded data, with secret-key accessible to authorized parties. See an illustration of our system in Figure 1.

Compatibility with industry's most commonly used architecture, tools and best-practices for storing and processing sensitive big data is attained by our system. For example we support the use of standardized encryption schemes for data protection in long-term storage (e.g. AES), native datatypes in datalake storage (e.g. double), and commonly used tools for computing servers and datalake (e.g. AWS EC2 instances with S3 storage). More broadly, incorporating our system into exiting pipelines requires minimal local expansion of current systems, supporting trans-

parent integration with existing applications and data processing flows e.g. by dynamically choosing whether data is retrieved in cleartext or in HE form. Importantly, our system does not only comply with existing architecture but simultaneously improves over it: to compute on data, current flows read it from datalake in exposed form, whereas our system supports oblivious data transformation from AES ciphertexts to CKKS (or other schemes of the user's choice) to be fed into the homomorphic computation with zero data exposure throughout the entire pipeline.

We ran extensive experiments with empirical results demonstrating that our solution achieves:

1. **Compact storage:** attaining zero overhead over using AES encryption to protect sensitive data; and attaining $10\times$ to $10^4\times$ improvement in storage size over directly storing CKKS encrypted data.
2. **Fast runtime:** attaining end-to-end storage plus oblivious retrieval runtime that is only twice the time of directly storing and retrieving CKKS encrypted data.

Furthermore, we demonstrate our system's support for flexible choice of the CKKS parameters by dynamically setting the degree of the cyclotomic polynomial, testing all values supported in [60], with essentially no degradation in our storage and runtime performance. Likewise, the data to be obliviously retrieved and how it is packed (batched) into CKKS ciphertexts can be flexibly set at the time of retrieval.

Concretely, on *two-million* data items, the storage size in our solution is 16 MB (cf. 168 MB in the baseline experiment of directly storing and retrieving batched CKKS ciphertexts with Z-standard library for compression), the storage runtime is 2s (cf. 8s), the oblivious retrieval runtime is 18s (cf. 2.5s), and communication size and time is 331 MB and 0.2s. The amortized performance per data item is therefore: 8 bytes of storage, 1μs and 9μs for storage and retrieval runtime respectively, and 165 bytes and 0.1μs communication.

We stress that the oblivious retrieval runtime is minor in comparison to the homomorphic computation to be applied on the encrypted data, e.g., when integrated with privacy preserving machine learning solutions. As a concrete example, *integrating our oblivious retrieval with homomorphic evaluation of a 64-tree random forest* on the retrieved cipheretexts using the non-interactive solution of [4] yields 10ms amortized time per data sample. See Table 3.

These results empirically demonstrate the practical usability of our system. We note that our implementation was single threaded and sequential, which does not utilize the fact that our solution is embarrassingly

parallelizable; extending our implementation to incorporate parallelization and streaming should gain a major further speedup.

Contribution 3: Secret sharing with fast homomorphic reconstruction over the reals. We present the first perfect secret sharing scheme supporting fast homomorphic reconstruction over the reals. The scheme offers 2-out-of-2 secret sharing for secrets in $[0, 1]$. The homomorphic reconstruction requires only additive homomorphism over the reals (as supported for example by CKKS), when given one encrypted share and the other in cleartext (more generally, reconstruction is a degree 2 polynomial over the reals, when both shares are the unknowns). In particular, the reconstruction algorithm does not require high degree computations such as reducing numbers modulo 1. In contrast, in all prior secret sharing schemes, the reconstruction algorithm is either computed over a finite field or ring (e.g., [61, 11, 14, 19, 20, 8, 49, 31, 26, 18]); or involves operations such as reducing numbers modulo one [10, 25] that are impractical for homomorphic computations even with state-of-the-art techniques [44]; or does not achieve perfect security [30, 62].

Theorem 1 (secret sharing (informal)). *There exists a 2-out-of-2 perfect secret sharing scheme for* $[0, 1]$ *whose reconstruction algorithm is a degree* 1 *polynomial over the reals when one of the two shares is the unknown. Consequently, reconstruction can be homomorphically evaluated (over one encrypted share) by any scheme supporting additive homomorphism over the reals.*

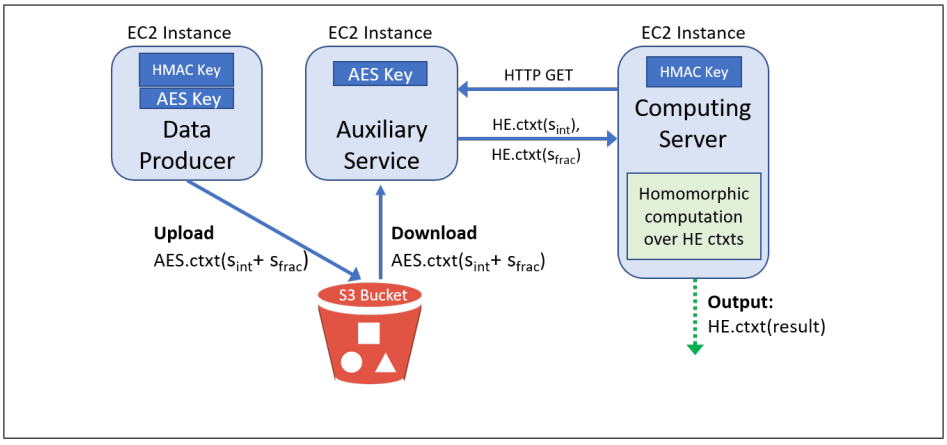


Fig. 1: Architecture and data flow

1.2 Overview of our Construction

We overview the main ideas and tools in our construction. See an illustration in Figure 1.

Store secret shares, homomorphically reconstruct to retrieve HE-ciphertexts. Our starting point is the following idea. To store data, first secret share it using a 2-out-of-2 secret sharing scheme (SSS), and store the 1st (resp. 2nd) share with access authorized to the computing (resp. auxiliary) server. To retrieve the data in HE-encrypted form, the auxiliary server encrypts its share by the HE and sends the ciphertext to the computing server; the computing server homomorphically reconstructs the data, using his cleartext share together with the received encrypted share, to obtain the data in HE-encrypted form.

To instantiate the above idea we require a "HE-friendly SSS" whose reconstruction algorithm can be efficiently computed over encrypted input. For HE schemes over finite rings such as Paillier [56] and BGV [15], additive secret sharing is friendly, as its reconstruction requires only additive homomorphism over the finite ring. In contrast, for HE over the reals, no friendly SSS was known prior to our work.

Friendly secret sharing for arithmetic over the reals. In our SSS, the key idea for achieving fast homomorphic reconstruction over the reals is to avoid the mod 1 step of [10, 25]. In [10, 25], sharing a number $x \in [0, 1)$ is by sampling a random $t \in [0, 1)$ and outputting shares $s_1 = t$ and $s_2 = x + t \bmod 1$; reconstructing is by outputting $s_1 + s_2 \bmod 1$. In our SSS for reals we replace the mod 1 step of [10, 25] with a masking of both the fractional and integral parts of $x + t$ as follows. To share a secret $x \in [0, 1]$ we sample independent uniformly random $t \in [0, 1)$ and $b \in \{0, 1\}$, outputting shares $s_1 = b + t$ and $s_2 = s_{int} + s_{frac}$ for $s_{int} = \lfloor x + t \rfloor + b \mod 2$ and $s_{frac} = x + t \mod 1$; whereas our reconstruction outputs $s_{frac} - t + (-1)^b s_{int} + b$ (all operations are computed over the reals).

The key property of our scheme is that, when given cleartext $b + t$ and encrypted s_{int}, s_{frac} as in our oblivious retrieval protocol, the reconstruction algorithm is a degree 1 polynomial in s_{int}, s_{frac}, and therefore it can be efficiently evaluated using any additive homomorphic encryption over the reals e.g. CKKS. Analogously, given cleartext $s_{int} + s_{frac}$ and encrypted $b, (-1)^b$ and t, the reconstruction algorithm is a degree 1 polynomial in the latter. The security of our SSS essentially follows by noticing that s_{frac} is a masking of x by adding to it a random $t \in [0, 1)$ and reducing modulo 1, and s_{int} is a masking of the integral part of $x + t$

by XOR-ing it with a random bit b. The key point however is that we never have to compute this reduction modulo 1 during reconstruction.

Storage optimized to zero overhead. To attain zero overhead over storing the data in cleartext, instead of storing both shares we store only the 2nd share (i.e., $s_2 = s_{int} + s_{frac}$), while generating the 1st share on-the-fly, pseudo-randomly, using a pseudorandom function f_k with key k accessible to the data producer and computing server, but not the auxiliary server. We require only a single persistent key k to be used in all store and retrieve sessions. This works because the 1st share in our SSS for reals is chosen uniformly at random and independently of the secret (likewise, in additive secret sharing over a finite ring). Since the share size in our SSS for reals (likewise, in additive secret sharing over a finite ring) has the same size as the cleartext data, the storage size is identical to storing the data in cleartext.

1.3 Other Related Work

Storage systems utilizing secret sharing. Secret sharing is used in the context of storage systems to ensure data availability and confidentiality, see a survey in [50] Section 4.6.9. In contrast, it has not been used for oblivious HE-retrieval prior to our work.

The shares size is a major consideration in the context of storage systems due to the high cost of storage. Secret sharing based storage solutions typically employ Krawczyk's computational secret sharing scheme [48] that has nearly optimal share size: the total size of the shares $\sum_{i=1}^{n} |s_i| = |x| + n|K|$ for $|x|$ the data size, $|K|$ the size of a symmetric encryption key freshly generated for each data item, and n the number of shares. All solutions use secret sharing over a finite field or ring (see a survey in [9]).

In contrast, in our work we propose a new secret sharing scheme for storage that is over real numbers. The total size of the shares in our scheme is $|x| + |K|$, consisting of the data x and a key K (cf. $|x| + 2|K|$ in [48]); moreover, the key is never revealed to the other party in our compact storage solution, and therefore we can use a single persistent key for all data (rather than requiring a fresh key to be generated and stored for each data item in [48]).

MPC utilizing secret sharing. Computing on secret shared data using an interactive protocol involving multiple rounds dates back to the circuit based methodology introduced in [37, 38, 21] and supported by a multitude of MPC frameworks such as ABY [29], SPDZ [27], Sharemind [14]

and more (see a survey in [40]). In all these protocols, the computation is over finite fields or rings, and the number of rounds is the mutiplicative depth of the circuit specifying the computation.

In contrast, in our solution secure computation requires only a single round of interaction for converting secret shares to data encrypted by homomorphic encryption, whereas the rest of the computation can be non-interactive via homomorphic evaluation on the encrypted data. Moreover, our scheme supports computation also over the reals (rather than only over a finite field or ring).

Mixed protocols. Mixed or hybrid protocols switch between several secure computation techniques and data representations in order to improve efficiency. In the ABY framework [29] the conversion is between Arithmetic (A) and Boolean (B) secret sharing and Yao's (Y) garbled circuits. In other works additive homomorphic encryption is employed to securely generate secret shares of Beaver multiplication triples (e.g. [28]) or of the data (e.g. [64]) by homomorphically evaluating the sharing algorithm of the secret sharing scheme. All these works use secret sharing over finite fields or rings.

In contrast, in our work we homomorphically evaluate the reconstruction algorithm of the secret sharing scheme (rather than its sharing algorithm); and the secret sharing scheme is over the reals (rather than a finite field or ring).

Homomorphic decryption for symmetric ciphers. Naehrig et al. [55] suggested transforming AES ciphertexts into HE ciphertexts via homomorphic evaluation of AES decryption when given as input a HE encryption of the AES decryption key. This was followed by a line of work for AES ciphers [34, 32] and other symmetric ciphers such as LowMC [6], Kreyvium [17], FLIP and FiLIP [53, 52], RASTA and MASTA [53, 39], and HERA [24] (see a survey [3]). However, this approach currently offers a running time that is not practical for handling massive amounts of data. Moreover, using the non standardized ciphers does not comply with industry practices of using AES.

In contrast, our solution is at least $10^4\times$ faster than the above works, albeit in the two-server model.

Paper Organization

The rest of this paper is organized as follows. Preliminary terminology and definitions appear in Section 2. The problem statement, including the system's description and defining the compact storage with oblivious

HE-retrieval primitive, in Section 3. Our constructions and theorems in Section 4. The implemented system and empirical evaluation in Section 5. Conclusions in Section 6.

2 Preliminaries

We use standard definitions for functions being *negligible* with respect to a system parameter λ called the *security parameter*, denoted $\mathsf{neg}(\lambda)$; similarly for *polynomial*, where ppt stands for *probabilistic polynomial time* in λ. We follow standard definitions for *probability ensembles* and *computationally indistinguishability*, denoted $\approx_c$; when two distributions are identical we denote this by $\equiv$. See the formal definitions in [45].

Definition 1 (secret sharing). *A* 2-out-of-2 secret sharing *scheme for* A *is a pair of* ppt *algorithms* $(\mathsf{Shr}, \mathsf{Rec})$ *such that:*

- Shr *is a randomized algorithm that given* $x \in \mathsf{A}$ *outputs a pair of shares* (s_1, s_2).
- Rec *is a deterministic algorithm that given a pair of shares* (s_1, s_2) *outputs an element in* A.

The correctness *requirement is that for all* $x \in \mathsf{A}$, $\mathsf{Rec}(\mathsf{Shr}(x)) = x$. *The (perfect)* security *requirement is that for every* $x, x' \in \mathsf{A}$ *and* $i \in \{1, 2\}$, *the following two distributions are identical:* $\{s_i\}_{(s_1,s_2)\leftarrow\mathsf{Shr}(x)} \equiv \{s'_i\}_{(s'_1,s'_2)\leftarrow\mathsf{Shr}(x')}$.

Definition 2 (homomorphic encryption (HE)). *A homomorphic public-key encryption (HE) scheme* $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *with message space* $\mathcal{M}$ *is a quadruple of* ppt *algorithms as follows:*

- Gen *(key generation) takes as input the security parameter* 1^λ, *and outputs a pair* (pk, sk) *consisting of a public key* pk *and a secret key* sk; *denoted:* $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$.
- Enc *(encryption) takes as input a public key* pk *and a message* $m \in \mathcal{M}$, *and outputs a ciphertext* c; *denoted:* $c \leftarrow \mathsf{Enc}_{pk}(m)$.
- Dec *(decryption) is a deterministic algorithm that takes as input a secret key* sk *and a ciphertext* c, *and outputs a decrypted message* m'; *denoted:* $m' \leftarrow \mathsf{Dec}_{sk}(c)$.
- Eval *(homomorphic evaluation) takes as input the public key* pk, *a circuit* $C\colon \mathcal{M}^\ell \to \mathcal{M}$, *and ciphertexts* $c_1, \ldots, c_\ell$, *and outputs a ciphertext* $\widehat{c}$; *denoted:* $\widehat{c} \leftarrow \mathsf{Eval}_{pk}(C; c_1, \ldots, c_\ell)$.

The correctness *requirement is that for every* (pk, sk) *in the range of* $\mathsf{Gen}(1^\lambda)$ *and every message* $m \in \mathcal{M}$*,*

$$\Pr[\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = m] \geq 1 - \mathsf{neg}(\lambda);$$

the scheme is $\mathcal{C}$-homomorphic *for a circuit family* $\mathcal{C}$ *if for all* $C \in \mathcal{C}$ *and for all inputs* $x_1, \ldots, x_\ell$ *to* C*, letting* $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ *and* $c_i \leftarrow \mathsf{Enc}_{pk}(x_i)$ *it holds that:*

$$\Pr[\mathsf{Dec}_{sk}(\mathsf{Eval}_{pk}(C; c_1, \ldots, c_\ell)) \neq C(x_1, \ldots, x_\ell)] \leq \mathsf{neg}(\lambda)$$

(where the probability is over all randomness in the experiment).

A HE scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ is *CPA-secure* if no ppt adversary $\mathcal{A}$ can distinguish between the encryption of two equal length messages x_0, x_1 of his choice; See the formal definition in [45].

Psedudorandom functions (PRF) We call an efficiently computable family of keyed functions $\mathcal{F} = \{f_k \colon \{0,1\}^* \to \mathrm{B}\}_{k \in \{0,1\}^\lambda, \lambda \in \mathbb{N}}$ *pseudorandom* if for all λ, a uniformly random function f_k from $\mathcal{F}$ s.t. $|k| = \lambda$ is computationally indistinguishable from a uniformly random function from the set of all functions having the same domain and range. See a formal definition in [45], Definition 3.25.

3 Problem Statement

Our goal is to attain compact storage with oblivious HE-retrieval that is compatible to existing system architecture in enterprises. To formally specify what this entails we first describe the relevant system components (Section 3.1) and then formally define the primitive of storage with HE-retrieval and its desired properties of oblivious retrieval and compact storage (Section 3.2). See illustrations in Figures 1 and 6.

3.1 System Description

The system entities most relevant to us are *data producers*, *computing server* and *auxiliary service*, denoted dataProd, compSrv and auxService, respectively. All entities have access to a storage repository and key management resources modeled as ideal functionalities, denoted $\mathcal{F}_{\text{storage}}$ and $\mathcal{F}_{\text{keys}}^{ids,\mathcal{E}}$ respectively. These resources may be accessed by all system entities, including ones beyond those considered here, using the same persistent resources in all sessions and protocols running in the ecosystem.

Data producers store data in the storage repository. There may be multiple data produces, and they may place data in storage over a period of time, possibly interleaved with retrieval requests from the computing server. The *computing server* retrieves from storage HE-ciphertexts for selected data items, where retrieval may involve an interactive protocol with the auxiliary service; the computing server may subsequently execute homomorphic computations such as privacy preserving machine learning over the retrieved encrypted data. The *auxiliary service* assists the computing server when called by it. The threat model assumes that the computing server and auxiliary service are non-colluding as formalized in Definition 3.

The storage functionality $\mathcal{F}_{\text{storage}}$ (Figure 2) supports data Upload and Download requests from parties.[3] Stored records are of the form $(index, value, id\_set)$ where $index$ is a unique record identifier, $value$ is the stored data, and id_set is the set of parties authorized to download the record. Access control can be enforced by storing encrypted data with symmetric key available to authorized parties.

The (asymmetric) key management functionality $\mathcal{F}_{\text{keys}}^{ids,\mathcal{E}}$ (Figure 3) supports key generation and key retrieval. The functionality is parameterized by a public-key encryption scheme $\mathcal{E}$ and a set of identities ids specifying which entities are authorized to access the secret key. Upon receiving a key retrieval request (Retrieve) from some party P, the functionality replies with the stored public key, and if P is an authorized party, i.e. $P \in ids$, then the reply includes also the secret key. The symmetric key management functionality is analogously defined.

Various company applications may be authorized to request computations on stored data and be permitted to access the result, albeit they might not have permission to access the raw data or may lack the computing resources required for the computation. The computing server may be called by such applications, specifying the data items to be retrieved and the computation to be homomorphically evaluated on them, and receiving from the computing server the encrypted result of the computation, where encryption is under the public key associated with the application identity in the key management functionality $\mathcal{F}_{\text{keys}}^{ids,\mathcal{E}}$. If the application is authorized to access the secret key then it may decrypt to obtain the cleartext result. In the following we focus on the storage and retrieval tasks while disregarding the integration with subsequent homomorphic computations, because the latter is straightforward by feeding the retrieved HE ciphertexts as input to the homomorphic computation. Furthermore, to

[3] We remark that records update is not included in the model.

simplify the presentation we hide details on the application requesting the computation by hard-wiring a single application's identity into the key management functionality; this can be extended in a straightforward manner to manage multiple keys for multiple identities.

Shared Functionality $\mathcal{F}_{\text{storage}}$

Shared functionality $\mathcal{F}_{\text{storage}}$ proceeds as follows:

Upload. Upon receiving from a party P a message:

$$(\mathsf{Upload}, index, value, id\_set)$$

record the tuple $(index, value, id\_set)$ (ignore if a record $(index, \cdot, id\_set)$ was previously recorded).

Download. Upon receiving from a party P_j a message:

$$(\mathsf{Download}, index)$$

output $(index, B)$ to P_j, where B is the set of all $value$ such that $(index, value, id\_set)$ was previously recorded by $\mathcal{F}_{\text{storage}}$ and id_set either contains P_j or is $\perp$.

Fig. 2: The storage functionality

Shared Functionality $\mathcal{F}_{\text{keys}}^{ids,\mathcal{E}}$

Parameterized by identities ids and a HE scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, shared functionality $\mathcal{F}_{\text{keys}}^{ids,\mathcal{E}}$ proceeds as follows:

Retrieve key. Upon receiving a ($\mathsf{Retrieve}$) message from a party P_j compute $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ and record it, if no record (pk, sk) exists. If $P_j \in ids$ return (pk, sk), otherwise return pk.

Fig. 3: The asymmetric keys functionality

3.2 Compact Storage with Oblivious Retrieval Primitive

The goal of *compact storage with oblivious HE-retrieval* is to eliminate the high storage overhead associated with storing data encrypted with HE. For this purpose we first define a primitive of *storage with HE-retrieval* supporting *store* and *retrieve* operations where retrieve must output the

data in HE-encrypted form. We then define the desired properties of obliviousness and compactness:

- *Obliviousness:* data secrecy must be preserved at all time, revealing no information on data content.
- *Compactness:* stored data may be maintained in any form, not necessarily encrypted by HE, with the goal of attaining lightweight storage. We call the storage *compact* if storage size is the same, up to a constant factor, as the data size.

When we want to explicitly specify the HE scheme $\mathcal{E}$ under which the retrieved data is encrypted, we use the name *compact storage with oblivious $\mathcal{E}$-retrieval*.

Definition 3 (compact storage with oblivious $\mathcal{E}$-retrieval). *Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be a HE scheme and A the data domain. A* storage with $\mathcal{E}$-retrieval for A *consists of two protocols $\pi = (\text{store}, \text{retrieve})$ executed on data $x \in \mathsf{A}$ to be stored and obliviously retrieved, a record identifier $index \in \{0,1\}^*$, and a security parameter λ, as follows.*

- store *is executed by* dataProd *on input $(x, index)$. During the execution,* dataProd *may perform some computations and may send $(\mathit{Upload}, index, \cdot, \cdot)$ to $\mathcal{F}_{\text{storage}}$ for some values and some identities (*dataProd *has no output).*
- retrieve *is executed by* compSrv *on input $index$, and by* auxService *that has no input. During the execution* compSrv *and* auxService *may interact, and both may send $(\mathit{Download}, index)$ to $\mathcal{F}_{\text{storage}}$ to download records, and may send $(\mathit{Retrieve})$ to $\mathcal{F}_{\text{keys}}^{ids,\mathcal{E}}$ to obtain the public key pk. The output of* compSrv *is a ciphertext c with respect to pk (whereas* auxService *have no output).*

We denote the output and view of compSrv *in an execution of π by $\mathsf{out}^{\pi}_{\mathsf{compSrv}}(x; index, \lambda)$ and $\mathsf{view}^{\pi}_{\mathsf{compSrv}}(x; index, \lambda)$ respectively (similarly for* auxService*), where the view consists of the party's input, randomness and received messages during the execution. We denote the values stored by $\mathcal{F}_{\text{storage}}$ in the execution by $\mathsf{storage}^{\pi}(x; index, \lambda)$.*

Correctness. *The correctness requirement is that* dataProd*,* compSrv *and* auxService *are ppt, and for any $x \in \mathsf{A}$, any $index$ s.t. at the time* store *was executed on $(x, index)$ a record $(index, \cdot, id\_set)$ with id_set containing $id_{\mathsf{auxService}}$ or id_{compSrv} has not been previously recorded in $\mathcal{F}_{\text{storage}}$,*[4] *and*

[4] The restriction on *index* is required by our modeling of $\mathcal{F}_{\text{storage}}$ with no updating of stored data; extensions are possible.

keys (pk, sk) *generated by* $\mathcal{F}_{\text{keys}}^{ids,\mathcal{E}}$ *it holds that:*

$$\Pr\left[\mathsf{Dec}_{sk}\left(\mathsf{out}_{\mathsf{compSrv}}^{\pi}\left(x; index, \lambda\right)\right) \neq x\right] \leq \mathsf{neg}(\lambda)$$

where the probability is taken over the random coins of π*.*

Obliviousness. *We say that* π *is* oblivious *if for every ppt* $\mathsf{compSrv}^*$ *and* $\mathsf{auxService}^*$*, and every ppt distinguisher* $\mathcal{D}$ *that chooses* $x_0, x_1 \in \mathsf{A}$ *s.t.* $|x_0| = |x_1|$ *and index, there exists a negligible function* $\mathsf{neg}(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$*, both the following holds:*

- $\Big|\Pr[\mathcal{D}(\mathsf{view}_{\mathsf{compSrv}^*}^{\pi}(x_0; index, \lambda)) = 1] - \Pr[\mathcal{D}(\mathsf{view}_{\mathsf{compSrv}^*}^{\pi}(x_1; index, \lambda)) = 1]\Big| \leq \mathsf{neg}(\lambda)$
- $\Big|\Pr[\mathcal{D}(\mathsf{view}_{\mathsf{auxService}^*}^{\pi}(x_0; index, \lambda)) = 1] - \Pr[\mathcal{D}(\mathsf{view}_{\mathsf{auxService}^*}^{\pi}(x_1; index, \lambda)) = 1]\Big| \leq \mathsf{neg}(\lambda)$

where the probability is over the random coins of all parties.

Compactness. *We say that* π *is* compact *(or,* t-compact*) if there exists a constant* $t > 0$ *s.t. for all* $x \in \mathsf{A}$*, index and* λ

$$|\mathsf{storage}^{\pi}(x; index, \lambda)| \leq t \cdot |x|$$

where $|z|$ *denotes the binary representation length of* z*.*

4 Our Compact Storage with HE-Retrieval

We present our compact storage with oblivious $\mathcal{E}$-retrieval. First we present our generic construction of a compact storage with oblivious $\mathcal{E}$-retrieval from any secret sharing scheme satisfying required properties that we define (Section 4.1). Next we present our secret sharing scheme for reals and prove it satisfies these properties (Section 4.2). We conclude by showing that instantiating our generic construction using our secret sharing scheme for reals yields a 1-compact storage with oblivious $\mathcal{E}$-retrieval over the reals (Corollary 2, Section 4.2).

4.1 Generic Construction from Secret Sharing

We present a generic construction of a t-compact storage with oblivious $\mathcal{E}$-retrieval from any secret sharing scheme satisfying the required properties specified below. See Figure 4 and Theorem 2 for the formal details of our generic construction, and its overview in Section 1.2.

Properties of secret sharing schemes: friendliness, random 1st share and compact 2nd share We first define the properties of 2-out-of-2 secret sharing schemes used in our generic construction: friendliness, random 1st share and compact 2nd share. We say that a secret sharing scheme $\mathcal{S}$ is *$\mathcal{E}$-friendly* with respect to a $\mathcal{C}$-homomorphic scheme $\mathcal{E}$, if homomorphic evaluation of its reconstruction algorithm with hardwired first share is in $\mathcal{C}$. We say that $\mathcal{S}$ has a *random 1st share* if its first share can be sampled uniformly at random independently of the secret. We say that $\mathcal{S}$ has a *t-compact 2nd share* if for every secret x, the 2nd share size is at most t times the size of x.

Our generic construction In Figure 4 we present our generic construction for storage with $\mathcal{E}$-retrieval, denoted $\pi = (\text{store}, \text{retrieve})$. In

Common parameters: A HE scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$, a $\mathcal{E}$-friendly 2-out-of-2 secret sharing scheme for a domain A with random 1st share, denoted $\mathcal{S} = (\mathsf{Shr}_1, \mathsf{Shr}_2, \mathsf{Rec})$, and a security parameter λ.

Parties and resources: The parties are dataProd, compSrv and auxService, all with access to the the (asymmetric) key management functionality $\mathcal{F}_{\text{keys}}^{ids,\mathcal{E}}$ and the storage functionality $\mathcal{F}_{\text{storage}}$. In addition, dataProd and compSrv (but not auxService) have access to a function f_k from a pseudorandom family $\mathcal{F} = \{f_k \colon \{0,1\}^* \to \mathcal{S}_1\}_{k \in \{0,1\}^\lambda, \lambda \in \mathbb{N}}$.

Inputs and outputs: The input of dataProd is $(x, index)$ for $x \in \mathsf{A}$; the input of compSrv is $index$ (other parties have no input). The output of compSrv is a ciphertext c (other parties have no output).

Storage protocol: store is executed by dataProd on input $(x, index)$, as follows:

1. Compute $s_1 \leftarrow f_k(index)$ and $s_2 \leftarrow \mathsf{Shr}_2(x, s_1)$
2. Send $(\mathsf{Upload}, index, s_2, \{id_{\mathsf{auxService}}\})$ to $\mathcal{F}_{\text{storage}}$

Retrieval protocol: retrieve is executed by compSrv on input $index$, as follows:

1. compSrv computes $s_1 \leftarrow f_k(index)$ and sends $index$ to auxService.
2. auxService sends $(\mathsf{download}, index)$ to $\mathcal{F}_{\text{storage}}$ to get s_2.
3. compSrv and auxService each send (Retrieve) to $\mathcal{F}_{\text{keys}}^{ids,\mathcal{E}}$ to get the public key pk.
4. auxService computes $c_2 \leftarrow \mathsf{Enc}_{pk}(s_2)$ and sends the ciphertext c_2 to compSrv.
5. Upon receiving c_2, compSrv computes $c \leftarrow \mathsf{Eval}_{pk}(\mathsf{Rec}_{s_1}; c_2)$ and outputs c.

Fig. 4: Our storage with oblivious HE-retrieval $\pi = (\text{store}, \text{retrieve})$ for A.

Theorem 2 we prove that our construction yields a t-compact storage with oblivious $\mathcal{E}$-retrieval for A.

Theorem 2 (compact storage w. oblivious retrieval). $\pi = (\text{store}, \text{retrieve})$ *in Figure 4 is a t-compact storage with oblivious $\mathcal{E}$-retrieval for* A*, whenever instantiated with λ, $\mathcal{E}$, $\mathcal{S}$ and f_k satisfying all the following:*

- *$\mathcal{E}$ is a CPA-secure HE scheme.*
- *$\mathcal{S}$ is a $\mathcal{E}$-friendly perfect secret sharing scheme for* A *with random 1st share and t-compact 2nd share.*
- *f_k is sampled uniformly at random from a pseudorandom function family $\mathcal{F} = \{f_k \colon \{0,1\}^* \to \mathcal{S}_1\}_{k \in \{0,1\}^\lambda, \lambda \in \mathbb{N}}$ s.t. $|k| = \lambda$.*

Proof. See Appendix A.

Compact storage with oblivious retrieval for finite rings We demonstrate how to instantiate our generic construction for finite rings $\mathbb{Z}_n$. Observe that *additive 2-out-of-2 secret sharing* over $\mathbb{Z}_n$ satisfies all the required properties (friendliness, random 1st share and compact 2nd share) as detailed next. Recall that in additive secret sharing, $\mathsf{Shr}(x)$ outputs uniformly random $s_1 \in \mathbb{Z}_n$ and $s_2 = x + s_1 \mod n$ and $\mathsf{Rec}(s_1, s_2)$ outputs $s_1 + s_2 \mod n$. Clearly, this has a random 1st share, 1-compact 2nd share, and is friendly with respect to any additive-homomorphic encryption over $\mathbb{Z}_n$. Together with Theorem 2 this implies the following corollary.

Corollary 1. *Figure 4 is a 1-compact storage with oblivious $\mathcal{E}$-retrieval for $\mathbb{Z}_n$, whenever instantiated with:*

- *$\mathcal{E}$ is a CPA-secure HE scheme supporting additive homomorphism over $\mathbb{Z}_n$, e.g., Paillier and BGV;*
- *$\mathcal{S}$ is the additive secret sharing scheme for $\mathbb{Z}_n$;*
- *f_k is sampled uniformly at random from a pseudorandom function family $\mathcal{F} = \{f_k \colon \{0,1\}^* \to \mathbb{Z}_n\}_{k \in \{0,1\}^\lambda, \lambda \in \mathbb{N}}$ s.t. $|k| = \lambda$ (the security parameter).*

Support for batched storage and retrieval. For simplicity of the presentation in Figure 4 we described storage and retrieval of a single data item. Nonetheless, our solution extends to support batched storage and retrieval of several data items at a time; moreover, the batched retrieval supports dynamic flexible choice of what data items to pack together in each ciphertext of the HE. Namely, store can be executed on input consisting of n pairs $(x_i, index_i)$; likewise, retrieve can retrieve multiple data items specified by their indices together with (optionally) a specification of which data items to pack together in each retrieved HE ciphertext

(batching pattern). We note that the batching pattern should of course comply with the upper bound on how many data items can be packed in each ciphertext.

The key for achieving the above is that auxService encrypts its shares while packing in each HE ciphertext the shares corresponding to the requested data items; likewise, the computing server encodes its plaintext shares while adhering to the requested batching pattern, and upon receiving the packed encrypted shares from auxService homomorphically evaluates the reconstruction algorithm in a single-instruction-multiple-data (SIMD) fashion.

4.2 Secret Sharing for Reals

We present our 2-out-of-2 secret sharing scheme for real numbers[5] in Figure 5 (see also the overview in Section 1.2), and prove it satisfies all properties required for our generic construction (as specified in Theorem 2).

Theorem 3 (secret sharing over the reals). *The scheme $\mathcal{S} = (\mathit{Shr}, \mathit{Rec})$ in Figure 5 is a secret sharing scheme for real numbers in $[0,1]$ (i.e. it is correct and secure). Furthermore, it has a random 1st share, 1-compact 2nd share, and it is $\mathcal{E}$-friendly for every HE scheme $\mathcal{E}$ with additive-homomorphism over the reals.*

Proof (Proof of Theorem 3, correctness). To prove correctness holds we show that for all $x \in \mathsf{A}$, $\mathsf{Rec}(\mathsf{Shr}(x)) = x$. Fix $x \in [0,1]$ and let $(s_1, s_2) \leftarrow \mathsf{Shr}(x)$. By definition of Shr there exists $b \in \{0,1\}$ and $t \in [0,1)$ such that $s_1 = b + t$ and $s_2 = (-1)^b (\lfloor x + t \rfloor - b) + x + t - \lfloor x + t \rfloor$. We inspect the steps of Algorithm Rec on input (s_1, s_2). In Step 1, Rec extracts from s_1 its integral and fractional parts, which by the definition of s_1 in $\mathsf{Shr}(x)$ are equal to the said values b and t respectively. In Step 2, Rec extracts from s_2 its integral and fractional parts, which by the definition of s_2 in $\mathsf{Shr}(x)$ are equal to $s_{int} = (-1)^b (\lfloor x + t \rfloor - b)$ and $s_{frac} = x + t - \lfloor x + t \rfloor$. In Step 3, Rec outputs $x' = s_{frac} - t + (-1)^b s_{int} + b$. Assigning the value $s_{int} = (-1)^b (\lfloor x + t \rfloor - b)$ we have that:

$$\begin{aligned} x' &= s_{frac} - t + (-1)^b((-1)^b (\lfloor x + t \rfloor - b)) + b \\ &= s_{frac} - t + (\lfloor x + t \rfloor - b) + b \end{aligned}$$

[5] The scheme takes secrets from the unit interval $[0,1]$; nonetheless, as noted in [25], composing a secret sharing scheme over $[0,1]$ with the bijection from all reals to this interval yields a secret sharing for arbitrary real numbers.

Assigning the value $s_{frac} = x + t - \lfloor x + t \rfloor$ we have that:

$$x' = x + t - \lfloor x + t \rfloor - t + (\lfloor x + t \rfloor - b) + b$$

which is in turn equal to x, and hence correctness holds.

Proof (Proof of Theorem 3, security). To prove that (perfect) security holds we show that for every $x \in \mathsf{A}$ and $i \in \{1, 2\}$, the distribution $\{s_i\}_{(s_1,s_2)\leftarrow\mathsf{Shr}(x)}$ is uniformly random in $[0, 2)$. The first share s_1 is the sum of independent and uniformly random $b \in \{0, 1\}$ and $t \in [0, 1)$, and so s_1 is uniformly random in $[0, 2)$. The second share s_2 is the sum of its integral and fractional parts s_{int} and s_{frac} respectively. We show below that the pair (s_{int}, s_{frac}) is uniformly random in $\{0, 1\} \times [0, 1)$, and so their sum $s_2 = s_{int} + s_{frac}$ is uniformly random in $[0, 2)$.

We show that the (s_{int}, s_{frac}) is uniformly random in $\{0, 1\} \times [0, 1)$. The integral part $s_{int} = (-1)^b (\lfloor x + t \rfloor - b)$ is the XOR of $\lfloor x + t \rfloor$ with a uniformly random bit b, and therefore s_{int} is a uniformly random bit. The fractional part $s_{frac} = x + t - \lfloor x + t \rfloor$ is equal to $(x + t)$ mod 1 for a uniformly random $t \in [0, 1)$, and therefore s_{frac} is uniformly random in $[0, 1)$. Moreover, since the randomness in s_{int} and s_{frac} emanates from the independent random variables b and t respectively, then s_{int} and s_{frac} are likewise independent, and so the pair (s_{int}, s_{frac}) is uniformly random in $\{0, 1\} \times [0, 1)$.

We conclude that s_1 and s_2 are uniformly random in $[0, 2)$, which in turn implies that for every $x, x' \in \mathsf{A}$ and $i \in \{1, 2\}$, the two distributions $\{s_i\}_{(s_1,s_2)\leftarrow\mathsf{Shr}(x)}$ and $\{s'_i\}_{(s'_1,s'_2)\leftarrow\mathsf{Shr}(x')}$ are identical, and therefore perfect security holds.

Proof (Proof of Theorem 3, random and compact shares). Our scheme has a random first share (specifically, the first share is $b + t$ for independent uniformly random $b \in \{0, 1\}$ and $t \in [0, 1)$). Furthermore, the second share s_2 is 1-compact for every input x, i.e. $|s_2| = |x|$, because $x \in [0, 1]$ and $s_2 \in [0, 2)$ are real numbers specified with the same precision.

Proof (Proof of Theorem 3, friendliness). By inspection, the reconstruction algorithm $\mathsf{Rec}_{b,t}(\cdot, \cdot)$ adds or subtracts the input values s_{int} and s_{frac}, the hardwired parameters b and t, and their product, but never multiplies together unknown input values. That is, $\mathsf{Rec}_{b,t}(\cdot, \cdot)$ computes a linear polynomial in its input (s_{int}, s_{frac}), and therefore $\mathsf{Rec}_{b,t}$ can be homomorphically evaluated by any additively homomorphic encryption scheme.

$\mathsf{Shr}(x)$. Given $x \in [0,1]$, sample independent and uniformly a random $b \in \{0,1\}$ and $t \in [0,1)$ and output a pair (s_1, s_2) defined by:

$$s_1 = b + t$$
$$s_2 = (-1)^b (\lfloor x + t \rfloor - b) + x + t - \lfloor x + t \rfloor$$

where all operations are computed over the reals.
In the following we denote the integral and fractional parts of s_2 by $s_{int} = (-1)^b (\lfloor x + t \rfloor - b)$ and $s_{frac} = x + t - \lfloor x + t \rfloor$, respectively.

$\mathsf{Rec}(s_1, s_2)$. Given shares $s_1, s_2 \in [0,2)$, do as follows:

1. Set $b \leftarrow \lfloor s_1 \rfloor$ and $t \leftarrow s_1 - b$
2. Set $s_{int} \leftarrow \lfloor s_2 \rfloor$ and $s_{frac} \leftarrow s_2 - s_{int}$
3. Output $s_{frac} - t + (-1)^b s_{int} + b$

Remark. *The pair (s_{int}, s_{frac}) (similarly, (b,t)) is an encoding of s_2 (similarly, s_1) by its integral and fractional parts. Denote by $\mathsf{Rec}_{b,t}(\cdot,\cdot)$ the reconstruction algorithm when b, t are parameters and (s_{int}, s_{frac}) is the input. Homomorphic evaluation of $\mathsf{Rec}_{b,t}(\cdot,\cdot)$ is supported by any encryption scheme that is additive-homomorphic over the reals.*

Fig. 5: Secret sharing for real numbers (all arithmetic is over the reals)

Remark 1 (Finite precision) *When implementing this scheme, the secret x is specified by finite precision (publicly known), and s_1 is sampled at the same precision. The analysis straightforwardly extend to this finite precision case. In particular, security holds as the shares are uniformly distributed in $[0,2)$ with the said precision; and correctness holds for all reals $x \in [0,1]$ and $s_1 \in [0,2)$, in particular for those captured by the finite precision.*

As a corollary of Theorem 2 and Theorem 3 we conclude that Figure 4, when instantiated with a HE scheme $\mathcal{E}$ over the reals (e.g. CKKS) together with our secret sharing scheme for reals, is a 1-compact storage with oblivious $\mathcal{E}$-retrieval for real numbers.

Corollary 2. *Figure 4 is a 1-compact storage with oblivious $\mathcal{E}$-retrieval for reals, whenever instantiated with:*

- *$\mathcal{E}$ is a CPA-secure HE scheme supporting additive homomorphism over the reals, e.g., CKKS.*
- *$\mathcal{S}$ is our secret sharing scheme for reals (Figure 5).*
- *f_k is sampled uniformly at random from a pseudorandom function family $\mathcal{F} = \{f_k \colon \{0,1\}^* \to [0,2)\}_{k \in \{0,1\}^\lambda, \lambda \in \mathbb{N}}$ s.t. $|k| = \lambda$ (the security parameter).*

5 Empirical Evaluation

We implemented a system running on AWS EC2 instances with AWS S3 storage realizing our compact storage with oblivious HE-retrieval over the reals, and ran extensive experiments demonstrating that our system has compact storage with zero overhead over storing AES encrypted data, and nearly optimal runtime complexity. Details on our system, experiments and results appears in Sections 5.1-5.3, respectively, further optimizations appear in Section 5.4.

5.1 The Implemented System

We implemented our compact storage with oblivious HE-retrieval (Figure 4), instantiated with: (1) CKKS [22] HE scheme using Microsoft SEAL version 3.6.2 [60]; (2) our secret sharing scheme for reals (Figure 5); and (3) a PRF based on HMAC with SHA-256 using OpenSSL version 1.1.1. The implementation is in C++, compiled with g++ version 9.3.0 and C++ standard 17. Everything is running on a Docker container, based on an Ubuntu 20.04.2 image.

We ran our implementation on AWS EC2 instances with AWS S3 storage. We use a separate AWS EC2 instance for each entity (dataProd, compSrv and auxService), all residing in the same AWS subnet. Our implementation is single threaded and runs on standard EC2 instances of type M5.2xlarge (32 GB RAM and up to 10 Gbps network bandwidth). The storage is in an AWS S3 bucket residing in the same AWS region as the EC2 instances. Access control is implemented by using the AES-GCM-256 symmetric key encryption to encrypt stored shares for the data under a secret-key accessible to dataProd and auxService, but not to compSrv. Communication between compSrv and auxService is via HTTP using Microsoft cpprestsdk library. See Figures 1 and 6.

5.2 Experiments

We measured storage size, runtime and communication in our system and compared performance to the baseline solution of storing and downloading the data directly in HE-encrypted form using Zstandard library compression.

We use the same parameters when executing both our system and the baseline solution, as follows. Both are implemented with CKKS in Microsoft SEAL as the HE scheme, using the same AWS S3 bucket for

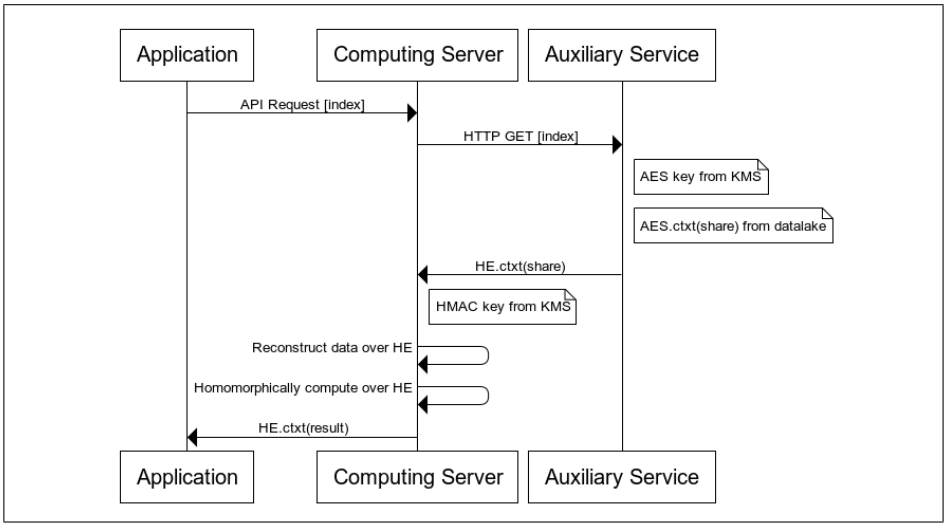


Fig. 6: System architecture

storage, and with security parameter 128 bits. The degree of the cyclotomic polynomial ranged over all values supported in SEAL: 8192, 16384 and 32768, with 30 bits of precision for the plaintext moduli at each level of homomorphic computation. The number *slots* of data items that can be packed in each ciphertext is half the degree of the cyclotomic polynomial. Performance is evaluated both in batched mode (packing *slots* data items in each ciphertext) and in un-batched mode (a single data item in each ciphertext). Data items are synthetically generated from $[0, 1]$ with finite precision, and represented as double-precision numbers according to the IEEE 754 standard [1]. The randomness in the secret sharing is taken with the same precision. The number of data items n ranges over the following values: 2, 4096, 16384, 98304, 507904 and 2031616.

Keys are kept in cache memory of the relevant party, and are reused across executions of the protocol. Each experiment is repeated 20 times, taking the average and standard-deviation.

We ran both end-to-end and microbenchmarks experiments. The end-to-end experiments compare the performance of our system vs. the baseline solution in:

- Storage size
- End-to-end runtime during store
- End-to-end runtime during retrieve

The microbenchmarks measure runtime in each computation step and the communication.

Table 1: Microbenchmarks to Figure 4 steps correspondence

Entity	**Data Producer**			**Computing Server**			**Auxiliary Server**			
Benchmark operation	**Share**	**AES encrypt**	**Upload**	**Process HTTP response**	**HMAC share derivation**	**Reconstruct**	**Download**	**AES decrypt**	**HE encrypt**	**Prepare HTTP response**
Fig. 4 step # or other details	store,1	access control	store,2	read stream & de-serialize	retrieve,1	retrieve,5	retrieve,2	access control	retrieve,4	serialize & write to stream

5.3 Results

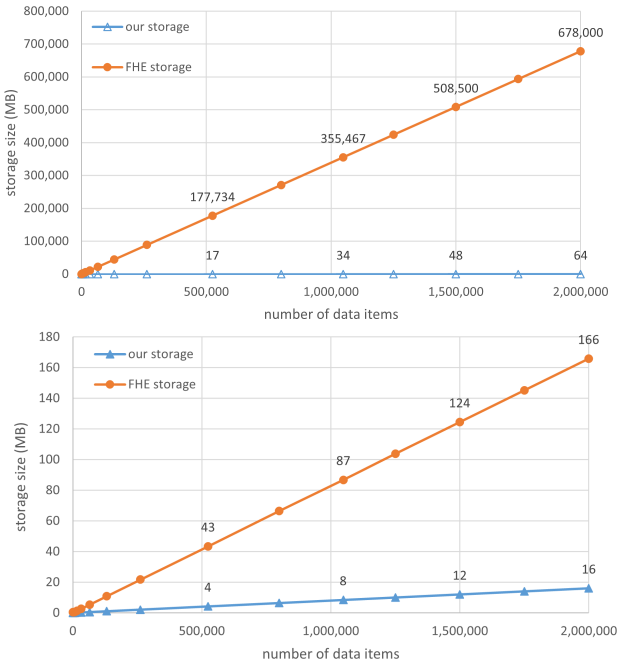
The empirical evaluation demonstrates our systems attains:

- Compact storage size: 10× better than the baseline in batched mode, and $10^4\times$ better if un-batched
- Fast end-to-end runtime: only twice the optimal time

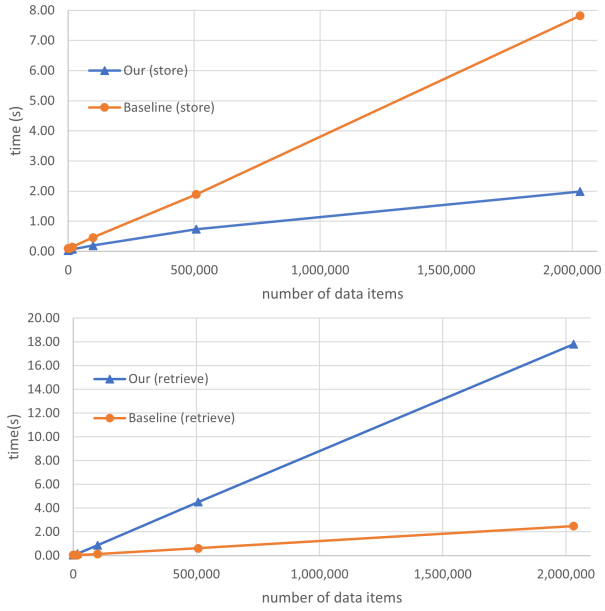
See Table 2 and Figures 7a-7b for our results in experiments with a degree 8192 cyclotomic polynomial. The experiments with degrees 16384 and 32768 cyclotomic polynomial show essentially no change in performance, provided full batching capacity is utilized (because the increase in complexity per ciphertext is fully compensated by the decrease in the number of ciphertexts due to *slots* being half that degree). Table 4 shows the breakdown of end-to-end storage and retrieval runtime measurements into specific operations and http communication complexity; See Table 1 for the correspondence to Figure 4.

Furthermore, we report the runtime for oblivious retrieval followed by homomorphic random forest evaluation, where the evaluation executes the non-interactive algorithm of [4] on encrypted data and cleartext forest consisting of 64 full binary trees of depth three. The number of data samples m in the homomorphic evaluation is: 128, 256, 1536, 8192 and 32768. The evaluation employs SIMD. Our results exhibit fast performance: 10ms amortized runtime per data sample for oblivious retrieval followed by the homomorphic random forest evaluation on the retrieved ciphertexts; See Table 3.

We point out that, unlike the baseline solution where one is forced to retrieve full batches, we support cherry picking of exactly the data items that are needed. Moreover, we support batching the retrieved data items according to the batching pattern specified at the time of retrieval, that can be tailored to best optimize the homomorphic computation, e.g., the random forest evaluation.



(a) Storage size (MB) in our solution vs. baseline, on data items encrypted one-by-one (top) or in batched form (bottom)



(b) End-to-end time (seconds) in our solution vs. baseline for storage (top) and retrieval (bottom)

Fig. 7: Size and time of our solution vs. baseline

Table 2: Storage size (MB) and runtime (seconds) in end-to-end experiments

Number of data items	Storage Size (MB)				Runtime (s)			
	Batched		Un-batched		Store		Retrieve	
	Ours	Baseline	Ours	Baseline	Ours	Baseline	Ours	Baseline
2	0.00003	0.3	0.00006	0.8	0.02	0.1	0.05	0.03
4,096	0.03	0.3	0.1	1,389	0.06	0.1	0.06	0.04
16,384	0.1	1.4	0.5	5,554	0.07	0.1	0.16	0.05
98,304	0.8	8	3	33,325	0.2	0.5	0.9	0.1
507,904	4	42	16	172,179	0.7	1.9	4.5	0.6
2,031,616	16	168	65	688,718	2.0	7.8	17.8	2.5

Table 3: Retrieve-then-evaluate runtime (seconds)

# Samples	Retrieve-then-Eval (s)
128	1.2
256	2.5
1,536	14.7
8,192	78.1
32,768	312.2

Table 4: Microbenchmarks

Number of data items	Runtime (seconds)										HTTP Communication	
	Data Producer			Computing Server			Auxiliary Server					
	Share	AES encrypt	Upload	Process HTTP response	HMAC share derivation	Reconstruct	Download	AES decrypt	HE encrypt	Prepare HTTP response	Size (MB)	Runtime (seconds)
2	0.00001	0.00001	0.02	0.01	0.00001	0.004	0.01	0.000003	0.01	0.01	0.7	0.001
4,096	0.003	0.00005	0.06	0.01	0.003	0.004	0.02	0.00001	0.01	0.009	0.7	0.001
16,384	0.01	0.0002	0.06	0.03	0.01	0.02	0.02	0.00004	0.06	0.04	2.7	0.001
98,304	0.08	0.001	0.1	0.2	0.07	0.09	0.03	0.0002	0.3	0.2	16	0.01
507,904	0.4	0.005	0.3	1.0	0.5	0.5	0.08	0.001	1.7	1.2	83	0.06
2,031,616	1.5	0.02	0.4	3.9	1.5	1.9	0.3	0.01	6.7	4.8	331	0.2

5.4 Optimizations: Parallel Computing and Streaming

The retrieval runtime in our solution can be significantly and easily optimized by parallelization. In our current implementation, all operations are performed sequentially. They can be fully parallelized so that each thread handles a single HE batch (for encryption) or HE ciphertext (for serialization, deserialization, reconstruction), that make up together about 78% of the retrieval runtime. Using AWS EC2 instances with to 96 vCPUs, we can reduce 78% of the current runtime by a factor of to 96.

Further optimization can be achieved by handling the data sent from Auxiliary Server to Computing Server as a stream. In the current implementation, Computing Server waits for the http response to be returned from Auxiliary Server before it starts to process it. In contrast, with streaming implementation, the Computing Server will process the HE ciphertexts as it gets them, while, in parallel, Auxiliary Server still generates the remaining ones. This will yield an overall running time that is the maximum of the Computing and Auxiliary servers runtime rather then their sum, which amounts to roughly 40% reduction in the total runtime.

6 Conclusions

We presented a compact storage with oblivious HE-retrieval solution that eliminates the high storage overhead associated with storing HE-ciphertexts. Our solution attains: compact storage, equal to storing AES ciphertexts; fast runtime, nearly as fast as directly storing and retrieving HE ciphertexts; dynamic control, at the time of retrieval, of the data items to be retrieved and the HE parameters and batching profile; compatibility to common industry's architecture tools and best-practices; and rigorous security analysis. As a central tool we introduce the first perfect secret sharing scheme with efficient homomorphic reconstruction over the reals. We implemented our solution into a system running on AWS EC2 instances and S3 storage. Further major performance speedup is available via parallelization and streaming. This gives a viable solution for use-cases requiring homomorphic computation on sensitive data in long-term storage.

References

1. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008*, pages 1–70, 2008.

2. Reform of EU data protection rules, May 2018.
3. A. Acar, H. Aksu, A. S. Uluagac, and M. Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (CSUR)*, 51(4):1–35, 2018.
4. A. Akavia, M. Leibovich, Y. S. Resheff, R. Ron, M. Shahar, and M. Vald. Privacy-preserving decision trees training and prediction. In *ECML/PKDD (1)*, pages 145–161, 2020.
5. A. Akavia, H. Shaul, M. Weiss, and Z. Yakhini. Linear-regression on packed encrypted data in the two-server model. In M. Brenner, T. Lepoint, and K. Rohloff, editors, *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography, WAHC@CCS 2019, London, UK, November 11-15, 2019*, pages 21–32. ACM, 2019.
6. M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for MPC and FHE. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 430–454. Springer, 2015.
7. A. B. Alexandru, M. Morari, and G. J. Pappas. Cloud-based MPC with encrypted data. In *57th IEEE Conference on Decision and Control, CDC 2018, Miami, FL, USA, December 17-19, 2018*, pages 5014–5019. IEEE, 2018.
8. M. Aliasgari, M. Blanton, Y. Zhang, and A. Steele. Secure computation on floating point numbers. In *NDSS*, 2013.
9. V. Attasena and N. Harbi. Secret sharing for cloud data security: a survey. *The VLDB Journal*, 26:657–681, 2017.
10. G. Blakley and L. Swanson. Security proofs for information protection systems. In *1981 IEEE Symposium on Security and Privacy*, pages 75–75. IEEE, 1981.
11. G. R. Blakley. Safeguarding cryptographic keys. In *Managing Requirements Knowledge, International Workshop on*, pages 313–313. IEEE Computer Society, 1979.
12. F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski. ngraph-he2: A high-throughput framework for neural network inference on encrypted data. In *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 45–56, 2019.
13. F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski. ngraph-he: a graph compiler for deep learning on homomorphically encrypted data. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*, pages 3–13, 2019.
14. D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations, esorics. *Google Scholar Google Scholar Digital Library Digital Library*, 2008.
15. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 309–325, 2012.
16. P. Bukaty. *The California Consumer Privacy Act (CCPA): An Implementation Guide*. IT Governance Publishing, 2019.
17. A. Canteaut, S. Carpov, C. Fontaine, T. Lepoint, M. Naya-Plasencia, P. Paillier, and R. Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. In T. Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 313–333. Springer, 2016.

18. O. Catrina. Efficient secure floating-point arithmetic using shamir secret sharing. In *ICETE (2)*, pages 49–60, 2019.
19. O. Catrina and S. De Hoogh. Improved primitives for secure multiparty integer computation. In *International Conference on Security and Cryptography for Networks*, pages 182–199. Springer, 2010.
20. O. Catrina and A. Saxena. Secure computation with fixed-point numbers. In *International Conference on Financial Cryptography and Data Security*, pages 35–50. Springer, 2010.
21. D. Chaum, C. Crépeau, and I. Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 11–19, 1988.
22. J. H. Cheon, A. Kim, M. Kim, and Y. S. Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, pages 409–437, 2017.
23. J. H. Cheon, D. Kim, Y. Kim, and Y. Song. Ensemble method for privacy-preserving logistic regression based on homomorphic encryption. *IEEE Access*, 6:46938–46948, 2018.
24. J. Cho, J. Ha, S. Kim, B. Lee, J. Lee, J. Lee, D. Moon, and H. Yoon. Transciphering framework for approximate homomorphic encryption. In M. Tibouchi and H. Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part III*, volume 13092 of *Lecture Notes in Computer Science*, pages 640–669. Springer, 2021.
25. B. Chor and E. Kushilevitz. Secret sharing over infinite domains. *Journal of Cryptology*, 6(2):87–95, 1993.
26. R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing. SPD$\mathbb{Z}_{2^k}$: Efficient mpc mod 2^k for dishonest majority. In *Annual International Cryptology Conference*, pages 769–798. Springer, 2018.
27. I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart. Practical covertly secure mpc for dishonest majority–or: breaking the spdz limits. In *European Symposium on Research in Computer Security*, pages 1–18. Springer, 2013.
28. I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.
29. D. Demmler, T. Schneider, and M. Zohner. Aby-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.
30. A. Dibert and L. Csirmaz. Infinite secret sharing–examples. *Journal of Mathematical Cryptology*, 8(2):141–168, 2014.
31. V. Dimitrov, L. Kerik, T. Krips, J. Randmets, and J. Willemson. Alternative implementations of secure real numbers. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 553–564, 2016.
32. Y. Doröz, Y. Hu, and B. Sunar. Homomorphic aes evaluation using ntru. *IACR Cryptol. ePrint Arch.*, 2014:39, 2014.
33. C. Gentry. *A fully homomorphic encryption scheme*. Stanford university, 2009.
34. C. Gentry, S. Halevi, and N. P. Smart. Homomorphic evaluation of the AES circuit. In R. Safavi-Naini and R. Canetti, editors, *Advances in Cryptology - CRYPTO*

2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, 2012.

35. I. Giacomelli, S. Jha, M. Joye, C. D. Page, and K. Yoon. Privacy-preserving ridge regression with only linearly-homomorphic encryption. In B. Preneel and F. Vercauteren, editors, *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*, volume 10892 of *Lecture Notes in Computer Science*, pages 243–261. Springer, 2018.
36. R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International conference on machine learning*, pages 201–210. PMLR, 2016.
37. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 307–328. 2019.
38. S. Goldwasser, M. Ben-Or, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computing. In *Proc. of the 20th STOC*, pages 1–10, 1988.
39. J. Ha, S. Kim, W. Choi, J. Lee, D. Moon, H. Yoon, and J. Cho. Masta: An he-friendly cipher using modular arithmetic. *IEEE Access*, 8:194741–194751, 2020.
40. M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic. Sok: General purpose compilers for secure multi-party computation. In *2019 IEEE symposium on security and privacy (SP)*, pages 1220–1237. IEEE, 2019.
41. E. Hesamifard, H. Takabi, M. Ghasemi, and R. N. Wright. Privacy-preserving machine learning as a service. *Proc. Priv. Enhancing Technol.*, 2018(3):123–142, 2018.
42. B. Jiang. Multi-key FHE without ciphertext-expansion in two-server model. *Frontiers Comput. Sci.*, 16(3):161809, 2022.
43. X. Jiang, M. Kim, K. Lauter, and Y. Song. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1209–1222, 2018.
44. C. S. Jutla and N. Manohar. Sine series approximation of the mod function for bootstrapping of approximate he. Cryptology ePrint Archive, Report 2021/572, 2021. `https://ia.cr/2021/572`.
45. J. Katz and Y. Lindell. *Introduction to modern cryptography*. CRC press, 2020.
46. A. Kim, Y. Song, M. Kim, K. Lee, and J. H. Cheon. Logistic regression model training based on the approximate homomorphic encryption. *BMC medical genomics*, 11(4):23–31, 2018.
47. M. Kim, Y. Song, S. Wang, Y. Xia, X. Jiang, et al. Secure logistic regression based on homomorphic encryption: Design and evaluation. *JMIR medical informatics*, 6(2):e8805, 2018.
48. H. Krawczyk. Secret sharing made short. In *Annual international cryptology conference*, pages 136–146. Springer, 1993.
49. T. Krips and J. Willemson. Hybrid model of fixed and floating point numbers in secure multiparty computations. In *International Conference on Information Security*, pages 179–197. Springer, 2014.
50. R. Kumar and R. Goyal. On cloud security requirements, threats, vulnerabilities and countermeasures: A survey. *Computer Science Review*, 33:1–48, 2019.

51. J. Liu, M. Juuti, Y. Lu, and N. Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 619–631, 2017.
52. P. Méaux, C. Carlet, A. Journault, and F. Standaert. Improved filter permutators for efficient FHE: better instances and implementations. In F. Hao, S. Ruj, and S. S. Gupta, editors, *Progress in Cryptology - INDOCRYPT 2019 - 20th International Conference on Cryptology in India, Hyderabad, India, December 15-18, 2019, Proceedings*, volume 11898 of *Lecture Notes in Computer Science*, pages 68–91. Springer, 2019.
53. P. Méaux, A. Journault, F. Standaert, and C. Carlet. Towards stream ciphers for efficient FHE with low-noise ciphertexts. In M. Fischlin and J. Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 311–343. Springer, 2016.
54. P. Mohassel and Y. Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 19–38. IEEE Computer Society, 2017.
55. M. Naehrig, K. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, page 113–124, New York, NY, USA, 2011. Association for Computing Machinery.
56. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*, pages 223–238. Springer, 1999.
57. M. S. Riazi, K. Laine, B. Pelton, and W. Dai. Heax: An architecture for computing on encrypted data. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1295–1309, 2020.
58. R. L. Rivest, L. Adleman, M. L. Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.
59. A. Sanyal, M. Kusner, A. Gascon, and V. Kanade. Tapas: Tricks to accelerate (encrypted) prediction as a service. In *International Conference on Machine Learning*, pages 4490–4499. PMLR, 2018.
60. Microsoft SEAL (release 3.6). `https://github.com/Microsoft/SEAL`, Nov. 2020. Microsoft Research, Redmond, WA.
61. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
62. K. Tjell and R. Wisniewski. Privacy in distributed computations based on real number secret sharing. *arXiv preprint arXiv:2107.00911*, 2021.
63. A. C.-C. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167. IEEE, 1986.
64. W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica. Helen: Maliciously secure coopetitive learning for linear models. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 724–738. IEEE, 2019.

A Proof of Theorem 2

In this section we prove Theorem 2.

Proof (Proof for Theorem 2). Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be a $\mathcal{C}$-homomorphic CPA-secure public key encryption scheme, let $\mathcal{S} = (\mathsf{Shr}_1, \mathsf{Shr}_2, \mathsf{Rec})$ be $\mathcal{E}$-friendly secret sharing scheme with random 1st share for domain A, and let $\lambda \in \mathbb{N}$ be a security parameter and f_k be a function sampled uniformly at random from a pseudorandom function family $\mathcal{F} = \{f_k \colon \{0,1\}^* \to \mathcal{S}_1\}_{k \in \{0,1\}^\lambda, \lambda \in \mathbb{N}}$ with $|k| = \lambda$.

The ppt complexity of dataProd, compSrv, and auxService follows from all components ($(\mathsf{Shr}_1, \mathsf{Shr}_2, \mathsf{Rec})$, f_k, Enc, and Eval) being ppt algorithms.

The correctness of π stems from the correctness and $\mathcal{E}$-friendliness of the secret sharing, and the $\mathcal{C}$-homomorphism of $\mathcal{E}$. More formally, fix some $x \in \mathsf{A}$, $index \in \{0,1\}^*$ and $\lambda \in \mathbb{N}$, and let $s_1 \leftarrow f_k(index)$ and $s_2 \leftarrow \mathsf{Shr}_2(s_1, x)$. The $\mathcal{E}$-friendliness together with $\mathcal{C}$-homomorphism of $\mathcal{E}$ guarantees that for every (pk, sk) in the range of $\mathsf{Gen}(1^\lambda)$ it holds that

$$\Pr\left[\mathsf{Dec}_{sk}\left(\mathsf{Eval}_{pk}\left(\mathsf{Rec}_{s_1}, \mathsf{Enc}_{pk}(s_2)\right)\right) \neq \mathsf{Rec}_{s_1}(s_2)\right] \leq \mathsf{neg}(\lambda) \tag{1}$$

In addition, the correctness of $\mathcal{S} = (\mathsf{Shr}_1, \mathsf{Shr}_2, \mathsf{Rec})$ and the range of $f_k \colon \{0,1\}^* \to \mathcal{S}_1$ guarantees correctness of the reconstructed value when Shr_1 is replaced with f_k, and therefore it follows that

$$\mathsf{Rec}_{s_1}(s_2) = \mathsf{Rec}(s_1, s_2) = x \tag{2}$$

Combining together Equations 1-2 we obtain

$$\Pr\left[\mathsf{Dec}_{sk}\left(\mathsf{Eval}_{pk}\left(\mathsf{Rec}_{s_1}, \mathsf{Enc}_{pk}(s_2)\right)\right) \neq x\right] \leq \mathsf{neg}(\lambda) \tag{3}$$

Moreover, the construction of π defines the output of compSrv to be

$$\mathsf{out}^{\pi}_{\mathsf{compSrv}}\left(x;\ index, \lambda\right) = \mathsf{Eval}_{pk}\left(\mathsf{Rec}_{s_1}, \mathsf{Enc}_{pk}(s_2)\right)$$

and hence implies together with Equation 3 the correctness of π, i.e.,

$$\Pr\left[\mathsf{Dec}_{sk}\left(\mathsf{out}^{\pi}_{\mathsf{compSrv}}\left(x;\ index, \lambda\right)\right) \neq x\right] \leq \mathsf{neg}(\lambda)$$

The t-compactness of π follows directly from the 2nd share t-compactness together with the construction in Figure 4, as for any input x only the second share s_2 is being stored and $|s_2| \leq t \cdot |x|$.

The obliviousness proof for any $\mathsf{compSrv}^*$ relies on its view being independent of x and only depending on $index$ (and presumably on the size of x), and hence on any equal size $x, x' \in [0,1]$ and every $index$ its view is indistinguishable. More formally, consider a construction $\bar{\pi}$ similar to π, where instead of Step 4 of retrieve in π the auxiliary service auxService encrypts a random $r \in \mathcal{S}_2$ s.t $|r| = |s_2|$. From the CPA-security

of $\mathcal{E}$ we obtain that for any $x \in \mathsf{A}$, $index \in \{0,1\}^*$ and $\lambda \in \mathbb{N}$ the following holds:

$$\mathsf{view}^{\pi}_{\mathsf{compSrv}^*}(x; index, \lambda) \approx_c \mathsf{view}^{\bar{\pi}}_{\mathsf{compSrv}^*}(x; index, \lambda) \tag{4}$$

In addition, the first share as derived in an execution of $\bar{\pi}$ depends only on $index$, i.e., $s_1 \leftarrow f_k(index)$. Since exactly the same s_1 is derived for any $x' \neq x$ the view of any $\mathsf{compSrv}^*$ is completely independent of x. Therefore, we obtain that for any $\lambda \in \mathbb{N}$, every ppt distinguisher $\mathcal{D}$ that chooses x, $x' \in \mathsf{A}$ s.t $|x| = |x'|$ and any $index$ the following holds:

$$\Big| \Pr[\mathcal{D}(\mathsf{view}^{\bar{\pi}}_{\mathsf{compSrv}^*}(x; index, \lambda)) = 1] - \\ \Pr[\mathcal{D}(\mathsf{view}^{\bar{\pi}}_{\mathsf{compSrv}^*}(x'; index, \lambda)) = 1] \Big| \leq \mathsf{neg}(\lambda)$$

Combining this with Equation 4 we obtain the desired.

For the obliviousness proof against any $\mathsf{auxService}^*$, first note that if π is executed on a previously used index then the view of $\mathsf{auxService}^*$ is completely independent of the current execution input data x, as nothing related to it was stored in $\mathcal{F}_{\text{storage}}$. Therefore we consider executions with a unique $index$.

Consider a version of π, denoted by $\bar{\pi}$, where f_k is replaced with Shr_1. That is, Step 1 of store in Figure 4 is replaced with $s_1 \leftarrow \mathsf{Shr}_1$ and $(\mathsf{Upload}, index, s_1, \{id_{\mathsf{compSrv}}\})$ to $\mathcal{F}_{\text{storage}}$ and Step 1 in retrieve is replaced with sending $(\mathsf{download}, index)$ to $\mathcal{F}_{\text{storage}}$ to get s_1. Since $\bar{\pi}$ is executed on a unique $index$ it holds that for any x and $index$ the distribution of shares (s_1, s_2) produced by $(\mathsf{Shr}_1, \mathsf{Shr}_2)$ is identical to the distribution produced by replacing Shr_1 with a uniformly, randomly sampled function R from the set of all functions $\{R : \{0,1\}^* \rightarrow \mathcal{S}_1\}$ and having $s_1 \leftarrow R(index)$. Moreover, the pseudorandomness property guarantees that for every x, the shares (s_1, s_2) produced by f_k are distributed computationally close to shares produced by R. Therefore, for any $\lambda \in \mathbb{N}$, $x \in \mathsf{A}$ and any $index$ the following views are indistinguishable,

$$\mathsf{view}^{\pi}_{\mathsf{auxService}^*}(x; index, \lambda) \approx_c \mathsf{view}^{\bar{\pi}}_{\mathsf{auxService}^*}(x; index, \lambda) \tag{5}$$

Next fix some value $y \in \mathsf{A}$ and consider a version of $\bar{\pi}$, denoted by $\bar{\pi}^y$, with a modified storage functionality $\mathcal{F}^y_{\text{storage}}$ that behaves as $\mathcal{F}_{\text{storage}}$ besides that on any download request from $\mathsf{auxService}^*$ with respect to any $index$ that is stored it computes $s_1 \leftarrow \mathsf{Shr}_1$ and $s_2 \leftarrow \mathsf{Shr}_2(s_1, y)$ and returns s_2. By the perfect security of $(\mathsf{Shr}_1, \mathsf{Shr}_2, \mathsf{Rec})$ and a hybrid argument it is guaranteed that for any $\lambda \in \mathbb{N}$, $x \in \mathsf{A}$ and every $index$

$$\mathsf{view}^{\bar{\pi}}_{\mathsf{auxService}^*}(x; index, \lambda) \equiv \mathsf{view}^{\bar{\pi}^y}_{\mathsf{auxService}^*}(x; index, \lambda) \tag{6}$$

Note that the view of any $\mathsf{auxService}^*$ in $\bar{\pi}^y$ is completely independent of the input x since in $\mathcal{F}^y_{\text{storage}}$ the reply of s_2 is derived from a fixed and independent value y. Therefore, we obtain that for any $\lambda \in \mathbb{N}$, every x, $x' \in \mathsf{A}$ s.t $|x| = |x'|$ and any $index$ the following holds:

$$\mathsf{view}^{\bar{\pi}^y}_{\mathsf{auxService}^*}(x; index, \lambda) \equiv \mathsf{view}^{\bar{\pi}^y}_{\mathsf{auxService}^*}(x'; index, \lambda)$$

Combining this with Equations 5 and 6 we obtain the desired, i.e., for any $\lambda \in \mathbb{N}$, every ppt distinguisher $\mathcal{D}$ that chooses x, $x' \in \mathsf{A}$ s.t $|x| = |x'|$ and any $index$,

$$\Big| \Pr[\mathcal{D}(\mathsf{view}^{\pi}_{\mathsf{auxService}^*}(x; index, \lambda)) = 1] - \Pr[\mathcal{D}(\mathsf{view}^{\pi}_{\mathsf{auxService}^*}(x'; index, \lambda)) = 1] \Big| \leq \mathsf{neg}(\lambda)$$

as desired.