Message Authentication Code with Fast Verification over Encrypted Data and Applications

Adi Akavia
University of Haifa
adi.akavia@gmail.com

Meir Goldenberg
University of Haifa
meirgold@hotmail.com

Neta Oren
University of Haifa
neta5128@gmail.com

Margarita Vald
Intuit Inc
margarita_vald@intuit.com

Abstract

In common data analytic scenarios, data is produced by a multitude of *data producers* (e.g., medical clinics), stored and maintained by some *data keeper* (e.g., a centralized repository), and substantial benefit can be gained from making data accessible to a variety of *data consumers* (e.g., researchers); however, making cleartext data accessible poses a privacy threat and may infringe on privacy regulation. Computing over data encrypted by fully homomorphic encryption (HE) enables providing privacy guarantee together with data mining utility. To ensure that correct insights are extracted, it is essential to guarantee *data authenticity*. In this work we present an authenticity proof for encrypted data:

(1) As a central tool we show how to modify a classical MAC based on universal hashing to introduce *the first MAC with fast homomorphic verification over the reals* (7.37 microseconds amortized runtime).
(2) We then utilize our MAC for guaranteeing data authenticity, for data provided by an untrusted data keeper in HE encrypted form. We implemented our solution, demonstrating *substantial efficiency improvements* over the prior art (Chatel et al. USENIX'21): improving the proof size and generation time by over $10^4\times$.

To demonstrate the usefulness of our homomorphic verification in realistic systems we implemented it in AWS EC2 with S3 storage, demonstrating it achieves practical performance for fetching and authenticating HE ciphertexts, as well as smooth integration with subsequent homomorphic evaluation of decision tree models.

Keywords

message authentication code, homomorphic encryption, authenticated storage

1 Introduction

In common data analytic scenarios, data is collected from multiple data producers, and significant insights can be gained from analyzing the data in its entirety. Examples include healthcare patients data that, if made accessible to medical researchers and companies, can be mined for the development of precision medicine; population data collected by a national census bureau that, when made available, allows gaining insights on population trends; companies collecting clients data to be accessed by their development teams to better the services they offer. In all these scenarios, data is produced by a multitude of *data producers* (e.g., medical clinics, client facing applications); data is stored and maintained by some *data keeper* (e.g., a centralized repository), possibly relying on third party infrastructure such as cloud storage; and substantial benefit can be gained from making data accessible to a variety of *data consumers* (e.g., researchers, data analytic teams).

Despite the potential gain in making data accessible, it is often not a possibility, e.g., when addressing sensitive data protected (as it should be) by privacy regulations (e.g., GDPR [28]) or when hindered by economic incentives to capitalize data for profit. The *secure computing* approach [31, 49] offers harnessing cryptographic techniques such as homomorphic encryption (HE) [30, 45] to compute on data while it is in encrypted form, revealing no information on the raw data (except the authorized output and what can be inferred from it). Namely, *secure computing enables mining data while keeping the data itself secret* [38]. In the context of scenarios as considered above, HE can be utilized as follows:

> The data keeper can provide data consumers with HE encrypted records; data consumers can compute over the encrypted data to produce an encrypted result (to be decrypted, if authorized).

Data consumers however should be concerned with regards to the authenticity of data received from the data keeper, who may be faulty or compromised. Indeed, holding large volumes of sensitive data makes the data keeper a lucrative attack target. Mitigating the data consumers concern is the focus of our paper. This entails ensuring authenticity of encrypted data received from the data keeper, i.e., ensuring they hold the requested data producers' data and are well-formed.

Ciphertexts authenticity, in some scenarios, can be guaranteed via the standard *Encrypt-then-Mac (EtM)* approach where data producers upload authenticated ciphertexts to the data keeper, and data consumers verify their authenticity. But the EtM approach is not applicable when the *data keeper is authorized to modify the ciphertexts*, e.g., for routine ciphertext updates that are common in data lifecycle, such as *re-encryption due to key rotation or policy changes*. Likewise, the EtM approach is not applicable when the data keeper stores data in cleartext, producing HE ciphertexts only when serving data consumers requests; this is because the data

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license visit https://creativecommons.org/licenses/by/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.
Proceedings on Privacy Enhancing Technologies 2025(4), 1092–1111
© 2025 Copyright held by the owner/author(s).
https://doi.org/10.56553/popets-2025-0173

keeper cannot authenticate on behalf of the data producers (as this would completely undermine unforgeability), and the data producers are not expected to be available for any task beyond uploading their data. In all such scenarios, the EtM approach is not applicable, because the data keeper is authorized to provide data in a form that differs from how data was uploaded by its producers.

The challenge is to guarantee that data remain intact, even when it is provided in encrypted form and where ciphertexts are created or processed by the data keeper. This can be achieved utilizing *zero knowledge proofs* [31, 32], by having the data keeper *prove* that the computations (e.g., encryption, re-encryption) were executed as prescribed and on the appropriate input and randomness, and the data consumers *verify* the proof; see an illustration in Figure 1. This could be instantiated using generic zero-knowledge proofs [20], but at a substantial complexity cost (e.g., proof generation time of 470*s* [20, Table 1, last row]).

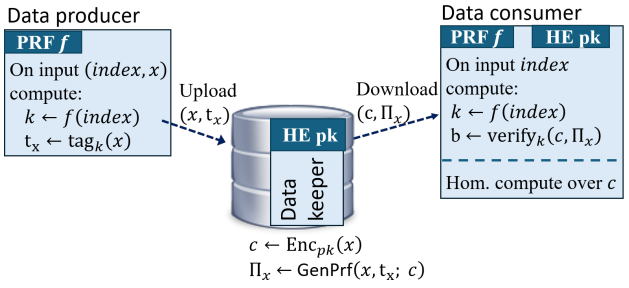

Figure 1: *Data producers* upload authenticated data to a centralized *data keeper*. The data keeper handles fetch requests, based on access permissions, providing HE encrypted data to *data consumers*, together with a proof for their validity. Data consumers verify, and compute over the encrypted data.

1.1 Our Contribution

In this work we show how to guarantee the authenticity of data provided in encrypted form, where the ciphertexts may have been created or modified by an untrusted data keeper. As a central tool we propose a message authentication code (MAC) with fast homomorphic verification. An open-source implementation is available in [6].

Contribution 1: MAC with fast homomorphic verification over the reals. For HE schemes supporting modular plaintext arithmetic (e.g., BGV [18], B/FV [17, 29], FHEW [26], TFHE [22]) MAC schemes with fast homomorphic tagging and verification exist; but not for schemes supporting plaintext arithmetic over the reals (e.g., the popular CKKS [21]). In this work we present:

> **Result 1:** *A MAC whose verification algorithm evaluates a linear polynomial over the reals.*

Thus supporting fast homomorphic verification over the reals.

Overview of our MAC. Our starting point is the classical (one-time) MAC based on the universal hashing $h_{\mathbf{k}}(\mathbf{x}) = \sum_{i=1}^{n} k_i x_i + k_0 \bmod p$, in which: the MAC key is uniform $\mathbf{k} \in \mathbb{Z}_p^{n+1}$; the tag for message $\mathbf{x} \in \mathbb{Z}_p^n$ is $h_{\mathbf{k}}(\mathbf{x})$; and verification of $(\mathbf{x}, y)$ accepts if-and-only-if $h_{\mathbf{k}}(\mathbf{x}) = y$. This MAC does not support fast homomorphic verification over the reals, because computing *modular reduction* over the reals and *testing equality*, both require evaluating a high degree polynomial.

In this work we modify the classical MAC as follows. To *avoid modular reduction* during verification, we do the following:

- Our tag augments the hash value $h_{\mathbf{k}}(\mathbf{x})$ —which is the *remainder* of $\sum k_i x_i + k_0$ in division by p— with a masking of the *quotient* in this division. The MAC key is extended to include the masking randomness. We *prove* that unforgeability holds via a reduction to the classical MAC.
- For masking we use a secret sharing scheme for $\mathbb{Z}_p$ whose reconstruction (we which use for unmasking) evaluates a *linear polynomial over the reals*. This secret sharing scheme is constructed via a lifting to $\mathbb{Z}_p$ of the secret sharing scheme for $[0, 1]$ in [8].
- Verification, given a message $\mathbf{x}$ and a tag consisting of a remainder and a masked quotient, operates as follows. The value $y := \sum k_i x_i + k_0$ is computed *over the reals (i.e., with no modular reduction)*. A (supposedly identical) value is computed from the tag by unmasking and combining the remainder y_r with the (unmasked) quotient y_q into: $y' := y_q \cdot p + y_r$. Note that computing y and y' is via evaluating *linear polynomials over the reals*. Verification accepts if-and-only-if $y = y'$.

Furthermore, to *avoid equality testing*, we replace it with *subtraction*, i.e., verification outputs $y - y'$, to be interpreted as accept if it equals 0 (reject otherwise). Combining all aforementioned modifications, ensures that verification evaluated a linear polynomial over the reals (in the message and tag as the unknowns).

Contribution 2: Ensuring data authenticity, in the ciphertext domain, against an untrusted data keeper. Next, we utilize our MAC as a central tool for instantiating the system in Figure 1. We implemented our solution, demonstrating significant complexity improvements compared to utilizing generic proofs [20] (see a detailed comparison in Table 1):

> **Result 2:** *We instantiate the system in Figure 1 utilizing our MAC as a central tool for ensuring authenticity over encrypted data; our instantiation exhibits amortized, per item, authenticity proof size and generation time of 13.94* bytes and 0.21μs, improving over the prior state-of-the-art [20] by 10^4 and 10^5, respectively.

The amortized verification time is 7.36μs, assuming CKKS is fully composable (see Definition 2.2) w.r.t. the verification circuit; otherwise it is 0.92*ms* (compare with 32*ms* in [20]). (The proof size and generation time are as stated above, regardless of whether CKKS is fully composable.)

Overview of our approach. We take a *Mac-then-Encrypt (MtE)* approach, as follows. Data producers generate authenticity tags directly on the data (in contrast to authenticating ciphertexts in the EtM approach). The data and tags are to be provided to data consumers in encrypted form. The data consumers homomorphically verify authenticity, to obtain an encrypted accept/reject outcome. Moreover, they can execute any homomorphic computation on the encrypted data to obtain an encrypted computation outcome. The

party receiving the verification and computation results in cleartext,[1] accepts them only if the verification output is "accept" (rejects, otherwise).

Remarks. (i) *On classical attacks on MtE, and their inapplicability to our settings.* In the context of authenticated encryption, there are well-known attacks on MtE [35] that leverage the accept/reject verification outcome to break the semantic security of the encryption scheme. These attacks are not applicable to our settings, because our data keeper knows the cleartext message, regardless of seeing verification outcomes. (ii) *On seeing decryption outcomes.* The inherent malleability of HE schemes implies insecurity against malicious adversaries who can access a decryption oracle as modeled in *CCA2-security*. In contrast, security against adversaries with access to decryptions of honest homomorphic computations, as modeled in *IND-CPA^D-security* [37], is guaranteed by the leading HE libraries (under standard cryptographic assumptions and with a bounded access to decryptions). We recommend employing such libraries in our solution, to ensure IND-CPA^D-security.

On the insufficiency of MAC alone, and our additional machinery. We note that our MAC, although a central tool, is not a sufficient one. We present an attack demonstrating this insufficiency:

> **Homomorphic verification alone may fail to guarantee unforgeability:** *We prove that for every unforgeable* $(\mathsf{Mac}, \mathsf{Vrf})$*, and HE* $\mathcal{E}$*, there exists a HE* $\widehat{\mathcal{E}}$*, and a ppt adversary* $\mathcal{A}$ *that produces ciphertexts* $(c_{\mathsf{msg}}, c_{\mathsf{tag}})$ *decrypting to* $(\mathsf{msg}, \mathsf{tag})$ *s.t.:*
>
> $$\begin{aligned} \mathsf{Vrf}(\mathsf{msg}, \mathsf{tag}) &= \mathsf{rej} \\ \widehat{\mathsf{Dec}}_{\widehat{sk}}\left(\widehat{\mathsf{Eval}}_{\widehat{pk}}(\mathsf{Vrf}; c_{\mathsf{msg}}, c_{\mathsf{tag}})\right) &= \mathsf{acc}. \end{aligned}$$
>
> *I.e., homomorphic verification with* $\widehat{\mathsf{Eval}}$ *fails to detect that* $(\mathsf{msg}, \mathsf{tag})$ *is a forgery.*

The attack, in a nutshell, is as follows. The public key of $\widehat{\mathcal{E}}$ is the public key of $\mathcal{E}$ augmented with a random ciphertext c^*; encryption and decryption ignore c^* and operate as in $\mathcal{E}$; the homomorphic evaluation $\widehat{\mathsf{Eval}}$ is similar to that of $\mathcal{E}$ except that on input containing c^* it always outputs $\mathsf{Enc}_{pk}(\mathsf{acc})$. $\widehat{\mathcal{E}}$ is semantically secure and correct, whenever $\mathcal{E}$ is. The adversary $\mathcal{A}$ sets $c_{\mathsf{msg}} := c^*$ (c_{tag} can be arbitrary); this guarantees that homomorphic verification $\widehat{\mathsf{Eval}}(\mathsf{Vrf}; c^*, c_{\mathsf{tag}})$ outputs a ciphertext that decrypts to acc.

We address this attack by incorporating machinery from [5] and [40] in our authenticity verification; details follow. Our starting point is to observe that the attack is rooted in the fact that HE schemes guarantee correct homomorphic evaluation only on ciphertexts generated by executing the scheme's encryption algorithm. In contrast, our attack feeds $\widehat{\mathsf{Eval}}$ with a ciphertext c^* taken from the public key, and not the output of encryption. Next, we prove that we can mitigate this attack, as well as any other attack, provided the HE scheme is fully composable [40], i.e., it guarantees correct homomorphic evaluation on arbitrary ciphertexts. In case the scheme is not fully composable, it can be transformed to fully composable (provided, it is equipped with bootstrapping, i.e., supports homomorphic computation of its decryption algorithm), by sanitizing [5] or bootstrapping [40] ciphertexts prior to homomorphic evaluation. Such a transformation increases the runtime of homomorphic verification by the time to sanitize or bootstrap its input ciphertexts; the proof (tag) size and generation time are unchanged. We note that full composability guarantees that ciphertexts are well-formed in the sense of ensuring correctness of homomorphic computation.

System implementation and applications. To demonstrate the usefulness of our homomorphic verification in realistic systems we implemented it in AWS EC2 instances (for data producer, keeper and consumer) with S3 bucket for storage. Our implementation demonstrates that our solution achieves practical performance for fetching and authenticating HE ciphertexts, with amortized per data item runtime of $0.92ms$ (on AWS EC2 c5.18xlarge with a single thread HE implementation). The bulk of this runtime is spent on bootstrapping CKKS ciphertexts for ensuring full composability w.r.t. our authentication, whereas the rest of the computation requires only additive homomorphism. Under the heuristic assumption that CKKS is fully-composable w.r.t. our computation, the runtime is considerably faster: amortized per data item runtime of $0.008ms$ (in an execution on 2M 64-bit data items; see Table 2).

To demonstrate that our system supports smooth integration with subsequent homomorphic computation we integrate it with the homomorphic decision tree evaluation in [7], using the open source code from [25], demonstrating it achieves high accuracy and fast runtime (see details in Section 5.3).

To further demonstrate the usability and flexibility of our system, we show how to integrate it with the *compact storage with HE-retrieval* solution of [8] for enhancing the latter with an authenticity guarantee (on top of their guarantee for *secrecy* against a malicious adversary who may corrupt the data keeper or consumer, but not both).

Notable, we present for these applications *generic* protocols (cf. Figures 5-6). Furthermore, we characterize sufficient properties of the MAC and HE scheme for use in our protocols (cf. Theorems 4.1-4.2). For example, our protocols can be instantiated with: (i) HE schemes supporting homomorphism over $\mathbb{Z}_p$ together with the classic universal hashing based MAC, as well as with (ii) HE schemes supporting homomorphism over the reals together with our MAC; see Remark 4.4.

1.2 Related Work

Our MAC (cf. Contribution 1) is an adaptation of the universal hashing based MAC of Figure 2 [47]. Other prior work on universal hashing based MAC include: UMAC [16, 36] and Poly1305 [44].

A work most closely related to our Contribution 2 is the work of Chatel et al. [20] that considers data integrity in the context of fetching HE ciphertexts from an untrusted data keeper. They consider 3-party settings with a data producer offloading cleartext data to an untrusted party analogous to our data keeper (called "user", there), who provides HE ciphertexts to an honest-but-curious server who executes homomorphic computations and wishes to be certain with regards to the authenticity of the underlying data. Their approach for ensuring data integrity is as follows: the data producer hashes the data (using SHA256 in their empirical evaluation) to produce a short digest, signs the digest using a digital signature, and sends the data and signed digest to the data keeper. Their

[1] Providing cleartext outcomes only for authorized parties and computations is treated as part of the outcome disclosure phase utilizing standard techniques.

data keeper then proves to the server that the provided ciphertexts are an honest encryption of the data signed by the data producer. They accomplish the latter using generic zero-knowledge proofs. Concretely, they employ ZKB++ [19] with BDOP commitment [12] to simultaneously verify a custom circuit that checks the integrity of the data and its correct encryption. Their proof size and generation time is 10,000× and 100,000× larger than ours.

The Akavia et al. [8] solution that we integrated into our system supports compact storage with fast and secure retrieval of HE ciphertexts, and data confidentiality guarantee against both data keeper and consumer (in the two server model, where the adversary can corrupt the data keeper or consumer, but not both). However, their protocol offers no authenticity guarantee. Our integrated system enhances the former with an authenticity guarantee by utilizing our MAC. Our integrated protocol can be viewed as an authenticated transciphering protocol, in the two server model.

Transciphering symmetric cipher to HE was first proposed in Naehrig et al. [43], with AES as the symmetric cipher. Subsequent works proposed replacing AES with other symmetric ciphers (e.g., LowMC [9], Chaghri [10], Rubato [33], Fasta [23], Elisabeth [24]) supporting faster transciphering runtime than from AES. Nonetheless, our solution is at least $10^4\times$ faster in runtime (more, if no bootstrapping is needed) and 10× lighter in storage space. Authenticated transciphering was first presented by Bendoukha et al. [13, 14], transciphering Grain128 to TFHE [22]. Their reported runtime is over $10^4\times$ slower than ours. Aharoni et al. [3] presented authenticated transciphering from Ascon and AES-GCM to CKKS. Our runtime is 4.33× faster, despite executing our system of a significantly weaker hardware (CPU vs. GPU). When comparing the monetary cost of the computation, our solution exhibits 55× reduction in cost.

Another relevant line of work addresses *homomorphic evaluation of hash functions* as means to provide data authenticity guarantee over encrypted data (and other applications). Homomorphic evaluation of SHA-256 hash function was first proposed by Mella and Susella [39] who proposed homomorphic evaluation of SHA-256 over data encrypted with BGV HE scheme. Bendoukha et al. [15] homomorphic evaluated, over data encrypted with TFHE, hash functions that were adapted from HE-friendly block ciphers (PRINCE, SPECK, SIMON, LowMC) with runtime ranging from 1.28 minutes to 17.32 minutes. Wei et al. [48] homomorphically evaluated SHA-256 and SM3 (motivated by their standardized status), on TFHE encrypted data, with runtime ranging from 7.41 minutes to 12 minutes per block (in the 1-core execution, and still over 1.5 minutes even when executing on 12-core machines). All cited results are for hashing 1-block. In contrast, our homomorphic MAC verification has amortized runtime of 7.37 microseconds.

Paper Organization

The rest of this paper is organized as follows: preliminary definitions and facts appear in Section 2; our MAC scheme in Section 3; its applications in Section 4; our empirical evaluation in Section 5; and conclusions in Section 6.

2 Preliminaries

Notations. We denote by $[0..p-1]$ the set of integers in the interval $[0, p-1]$. We denote vectors by boldface letters, and index them starting from 1 unless explicitly specified otherwise, e.g., $\mathbf{x} = (x_1, \ldots, x_n)$. We denote $[n] = \{1, \ldots, n\}$. Single-Instruction-Multiple-Data (SIMD) operations operate on vectors entry-by-entry, e.g., SIMD multiplication of $\mathbf{x}$ and $\mathbf{y}$ produces their point-wise product $(x_1y_1, \ldots, x_ny_n)$. The operations returning the quotient and remainder in integer division are denoted by $\div$ and mod, respectively; and we extend them to be applied to vectors of integers entry-by-entry, i.e., $\mathbf{x} \div p = (x_1 \div p, \ldots, x_n \div p)$ and $\mathbf{x} \bmod p = (x_1 \bmod p, \ldots, x_n \bmod p)$. We use $|z|$ to denote the binary representation length of z. We denote by $s \leftarrow_R S$ a uniformly random sample s from a set S.

We use standard definitions for functions being *negligible* and *polynomial* with respect to a system parameter λ called the *security parameter*; for *probability ensembles* and for *computationally indistinguishability*; see formal definitions, e.g., in [34]. We use the notation $\mathsf{neg}(\lambda)$ to denote a function that is negligible in λ; use the shorthand notation "ppt" for algorithms that run in *probabilistic polynomial time* (in λ); use $\equiv$ (resp. $\approx_c$) to denoted that two distributions are identical (resp. computationally indistinguishable).

We defer to Appendix B the formal definitions of MAC, homomorphic encryption, pseudo-random functions, and secret sharing. We bring here only the construction of the affine MAC which we adapt to our settings, the definition of full composability, and establish some terminology we use in the context of secret sharing.

Constructing MAC from universal hashing. Wegman and Carter [47] presented a one-time (information theoretically) unforgeable MAC construction using a family of *(strongly) universal hash* functions $\{h_k : \mathsf{A} \to \mathsf{B}\}_{k\in\mathcal{K}}$, i.e., functions that satisfy the following: for all distinct messages $x, x' \in \mathsf{A}$ and all $y, y' \in \mathsf{B}$,

$$\Pr[h_k(x) = y \ \wedge \ h_k(x') = y'] = \frac{1}{|\mathsf{B}|^2}$$

(where the probability is over the choice of $k \in \mathcal{K}$). Their MAC construction is as follows: Gen samples a uniformly random $k \in \mathcal{K}$; $\mathsf{Mac}_k(x)$ outputs $h_k(x)$; and $\mathsf{Vrf}_k(x, y)$ outputs acc if-and-only-if $h_k(x) = y$ (rej otherwise). Unforgeability follows by the fact that even after seeing one message-tag pair: $(x, h_k(x))$, still the correct tag $h_k(x')$ for any other message x' is uniformly distributed in B (from the point of view of an adversary), and therefore the adversary's probability of forging is the probability of randomly guessing the tag, i.e., probability $1/|\mathsf{B}|$. So the scheme is unforgeable as long as $1/|\mathsf{B}|$ is negligible in the security parameter. (A formal proof can be found, e.g., in [34, Theorem 4.25].)

The above can be instantiated with any universal hashing. Figure 2 presents an instantiation with the universal hash function $h : \mathbb{Z}_p^{n+1} \times \mathbb{Z}_p^n \to \mathbb{Z}_p$ (for p a prime) defined over keys $\mathbf{k} = (k_0, \ldots, k_n)$ and messages $\mathbf{x} = (x_1, \ldots, x_n)$ by:

$$h_{\mathbf{k}}(\mathbf{x}) = k_0 + \sum_{i=1}^{n} k_i x_i \bmod p. \tag{1}$$

Theorem 2.1. *Figure 2 is a $\frac{1}{p}$-one-time unforgeable MAC.*

A HE is *fully composable* if $\mathcal{C}$-homomorphism (cf. Definition B.3) holds not only w.r.t well-formed ciphertexts but w.r.t all ciphertexts

(1) $\mathsf{Gen}^{n,p}()$. Output $\mathbf{k} = (k_0, \ldots, k_n) \leftarrow_R \mathbb{Z}_p^{n+1}$
(2) $\mathsf{Mac}^{n,p}_{\mathbf{k}}(\mathbf{x})$, given $x \in \mathbb{Z}_p^n$, output tag $:= k_0 + \sum_{i=1}^n k_i x_i \bmod p$.
(3) $\mathsf{Vrf}^{n,p}_{\mathbf{k}}(\mathbf{x}, y)$, given $(x, y) \in \mathbb{Z}_p^n \times \mathbb{Z}_p$, output acc iff $y = k_0 + \sum_{i=1}^n k_i x_i \bmod p$.

Figure 2: MAC for $\mathbb{Z}_p^n$ via universal hashing

[40]. This property can be guaranteed by bootstrapping prior to homomorphic evaluation [40].

Definition 2.2 (Fully composable [40]). *A HE scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ is* fully composable *with respect to a circuit family C if for all circuits $C \in C$ and all $\mathbf{c} := (c_1, \ldots, c_\ell)$ where c_i are in the ciphertext space of $\mathcal{E}$ and ℓ in the number of inputs to C, letting $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ it holds that:*

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

Definition 2.3 (secret sharing properties (adapted from [8])). *Let $\mathcal{S} = (\mathsf{Shr}, \mathsf{Rec})$ be a secret sharing scheme for a domain A. Denote the set of possible values for share $i = 1, 2$, by $\mathcal{S}_i = \{s_i \mid x \in \mathsf{A}, (s_1, s_2) \leftarrow \mathit{Shr}(x)\}$. For every $s_1 \in \mathcal{S}_1$, denote by Shr_{s_1} and Rec_{s_1} the algorithms Shr and Rec respectively with the 1st share hardwired to be s_1. We say that $\mathcal{S}$ has a* random 1st share *if for all $x \in \mathsf{A}$,*

$$\{(s_1, s_2) \leftarrow \mathsf{Shr}(x)\} \equiv \{(s_1, s_2) \mid s_1 \leftarrow_R \mathcal{S}_1, s_2 \leftarrow \mathsf{Shr}_{s_1}(x)\}.$$

We say that $\mathcal{S}$ has a uniform ith share *($i = 1, 2$) if for all $x \in \mathsf{A}$, $\{s_i\}_{(s_1,s_2)\leftarrow\mathsf{Shr}(x)} \equiv \{s_i\}_{s_i \leftarrow_R \mathcal{S}_i}$. Let $\mathcal{E}$ be a C-homomorphic public-key encryption scheme. We say that $\mathcal{S}$ is $\mathcal{E}$-*friendly *if the following holds: $\mathit{Rec}_{s_1} \in C$ for all $s_1 \in \mathcal{S}_1$, and $\mathcal{S}_2$ is in the message space of $\mathcal{E}$.*

3 Our Message Authentication Code

We present our Message Authentication Code (MAC) in Figure 4. Its key novel property is that verification is by computing a linear polynomial over the reals (with no modular reduction). This property makes our MAC HE-friendly for HE schemes supporting additive homomorphism over the reals, e.g., CKKS [21]. In the following we present our MAC (Section 3.1), and its HE-friendliness (Section 3.2).

3.1 MAC with Linear Verification over Reals

We present our MAC in Figure 4 and Theorem 3.2.

Our MAC employs (as a black-box) a secret sharing scheme for integer vectors with the following properties: a random 1st share, a uniform 2nd share, and reconstruction by evaluating a linear polynomial over the reals. Our MAC can employ any secret sharing scheme with these properties; we present one such scheme –which is an adaptation of the secret sharing scheme for $[0, 1]$ in [8] to our domain of integer vectors– in Figure 3 and Theorem 3.1.

Theorem 3.1 (secret sharing). *The scheme $(\mathit{Shr}^{\ell,p}, \mathit{Rec}^{\ell,p})$ in Figure 3 is a 2-out-2 perfect secret sharing for $[0..p-1]^\ell$, provided $\mathcal{S}_1 \leftarrow_R \{0,1\}^\ell \times [0..p-1]^\ell$, with a random 1st share, uniform 1st and 2nd shares, and shares size $|\mathbf{s_i}| = \ell \cdot (\lfloor \log_2 p \rfloor + 2)$. Moreover, $\mathit{Rec}^{\ell,p}$ evaluates a linear polynomial over the reals (in the 2nd share as the unknown).*

$\underline{\mathsf{Shr}^{\ell,p}_{\mathbf{s_1}}(x)}$. Given $\mathbf{s_1} := (\mathbf{b}, \mathbf{t}) \in \{0,1\}^\ell \times [0..p-1]^\ell$, and number x represented in base-p as $(x_1, \ldots, x_\ell) \in [0..p-1]^\ell$, output $\mathbf{s_2} := (\mathbf{s}^{\oplus}_{\mathsf{q}}, \mathbf{s}_{\mathsf{r}}) \in \{0,1\}^\ell \times [0..p-1]^\ell$ defined by:

$$s^{\oplus}_{\mathsf{q},j} := \left\lfloor (x_j + t_j) \div p \right\rfloor + b_j \bmod 2$$
$$s_{\mathsf{r},j} := x_j + t_j \bmod p$$

$\underline{\mathsf{Rec}^{\ell,p}_{\mathbf{s_1}}(\mathbf{s_2})}$. Given $\mathbf{s_1} := (\mathbf{b}, \mathbf{t})$ and $\mathbf{s_2} := (\mathbf{s}^{\oplus}_{\mathsf{q}}, \mathbf{s}_{\mathsf{r}})$, both in $\{0,1\}^\ell \times [0..p-1]^\ell$, output a number x in base-p representation $(x_1, \ldots, x_\ell)$ defined by:

$$x_j := \left[(-1)^{b_j}(s^{\oplus}_{\mathsf{q},j} - b_j)\right] \cdot p + s_{\mathsf{r},j} - t_j$$

(with arithmetic over the reals)

Figure 3: Secret sharing for numbers given in base-p representation $(x_1, \ldots, x_\ell)$. We denote by "$u \div p$" and "$u \bmod p$" the quotient and remainder in division of a number u by p (where numbers in base p are interpreted as integers).

$\underline{\mathsf{Gen}^{n,p}()}$. Output $(\mathbf{k}, \mathbf{s_1}) \leftarrow_R [0..p-1]^{n+1} \times \mathcal{S}_1$ (where entries of $\mathbf{k}$ and $\mathbf{s_1}$ are indexed from 0 and 1, respectively).

$\underline{\mathsf{Mac}^{n,p}_{\mathbf{k},\mathbf{s_1}}(x)}$.. Given a key $(\mathbf{k}, \mathbf{s_1}) \in [0..p-1]^{n+1} \times \mathcal{S}_1$ and a number in base-p representation $(x_1, \ldots, x_n) \in [0..p-1]^n$, do:
(1) $y := k_0 + \sum_{i=1}^n k_i x_i$ and $(y_{\mathsf{q}}, y_{\mathsf{r}}) := (y \div p, y \bmod p)$
(2) $\mathbf{s_2} \leftarrow \mathsf{Shr}^{\ell,p}_{\mathbf{s_1}}(y_{\mathsf{q}})$
(3) Output tag $:= (\mathbf{s_2}, y_{\mathsf{r}})$

$\underline{\mathsf{Vrf}^{n,p}_{\mathbf{k},\mathbf{s_1}}(x, (\mathbf{s_2}, y_{\mathsf{r}}))}$.. Given a key $(\mathbf{k}, \mathbf{s_1}) \in [0..p-1]^{n+1} \times \mathcal{S}_1$, a number in base-$p$ representation $(x_1, \ldots, x_n) \in [0..p-1]^n$, and a tag $(\mathbf{s_2}, y_{\mathsf{r}}) \in \mathcal{S}_2 \times [0..p-1]$, do:
(1) $y_{\mathsf{q}} \leftarrow \mathsf{Rec}^{\ell,p}_{\mathbf{s_1}}(\mathbf{s_2})$ and $y := y_{\mathsf{q}} \cdot p + y_{\mathsf{r}}$
(2) $y' := k_0 + \sum_{i=1}^n k_i x_i$
(3) Output diff $:= y' - y$, interpreted as acc if diff $= 0$

Figure 4: MAC for numbers given in base-p representation $(x_1, \ldots, x_n)$. The MAC employs (as a black box) a 2-out-of-2 perfect secret sharing scheme $(\mathbf{Shr}^{\ell,p}_{\mathbf{s_1}}, \mathbf{Rec}^{\ell,p}_{\mathbf{s_1}})$ for $[0..p-1]^\ell$, where $\ell := \left\lfloor \log_p(np^2) \right\rfloor$, that has a random 1st share in $\mathcal{S}_1$ and a uniform 2nd share in $\mathcal{S}_2$; e.g., the scheme specified in Figure 3. We denote by "$u \div p$" and "$u \bmod p$" the quotient and remainder in division of a number u by p (where numbers in base p are interpreted as integers).

Proof. The proof is straightforward from [8, Theorem 3.2]. □

Now, we are ready to present our MAC – see Figure 4.

Theorem 3.2 (MAC). *For every positive integer n and prime p, the scheme $(\mathsf{Gen}^{n,p}, \mathsf{Mac}^{n,p}, \mathsf{Vrf}^{n,p})$ in Figure 4 satisfies the following.*

- *It is a $\frac{1}{p}$-one-time unforgeable MAC.*
- *The tag is in $\mathcal{S}_2 \times [0..p-1]$. The tag's size is $O(\log_2(np^2))$, when instantiated with the secret sharing scheme of Figure 3.*

- $\mathsf{Vrf}^{n,p}$ *evaluates a linear polynomial over the reals (in message and tag as the unknowns).*
- *For every* $x_1, x_2 \in [0..p-1]^n$ *it holds that:*

$$\{\mathsf{Mac}^{n,p}_{\mathbf{k},\mathbf{s_1}}(x_1)\}_{(\mathbf{k},\mathbf{s_1})\leftarrow \mathsf{Gen}^{n,p}()} \equiv \{\mathsf{Mac}^{n,p}_{\mathbf{k},\mathbf{s_1}}(x_2)\}_{(\mathbf{k},\mathbf{s_1})\leftarrow \mathsf{Gen}^{n,p}()}$$

Proof. The proof appears in Appendix C.1.2. □

Remark 3.3 (Extension to reals in fixed-point representation). *The schemes in Figures 3-4 can be applied on any number* $(x_1, \ldots, x_n)$ *in base-p representation, e.g., integers* $x = \sum_{i=1}^{n} x_i p^{i-1}$*; real numbers in fixed-point precision* $x = \Delta^{-1} \cdot \sum_{i=1}^{n} x_i p^{i-1}$ *where the scale* Δ *specified by the format. More generally, they can be applied to any data (not necessarily a number) that is encoded to* $[0..p-1]^n$*.*

Remark 3.4 (Amplification to negligible forging probability). *To guarantee that the forging probability is negligible in the security parameter* λ*, either set* $p \geq 2^{\lambda}$ *or execute* $k := \lambda / \lceil \log_2 p \rceil$ *independent repetition (as detailed in Figure 10 in Appendix C.1.1). The key generation in the amplified scheme is denoted by* $\mathsf{Gen}(1^{\lambda})$*.*

3.2 HE-Friendliness of our MAC

In this section we first formally define HE-friendliness for MAC schemes, and show that our MAC is HE-friendly. Next, we extend the correctness and unforgeability definitions to verification over encrypted data (HE-correctness and HE-unforgeability). We then prove that our MAC is HE-correct. Finally we prove that our MAC HE-unforgeable when evaluated with an HE scheme that is fully-composable w.r.t its verification algorithm (cf. Definition 2.2).

We call a MAC *HE-friendly*, if its verification algorithm can be homomorphically evaluated over encrypted message and tag.

Definition 3.5 (HE-friendly MAC). *Let* $\mathcal{Q} = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrf})$ *be a MAC for message space* A*; denote its tag space by* Tgs*. Let* $\mathcal{E}$ *be a* C*-homomorphic public-key encryption scheme. We say that* $\mathcal{Q}$ *is* $\mathcal{E}$-friendly*, if*

(1) $\mathsf{Vrf}_{\mathsf{k}}(\cdot,\cdot) \in C$ *for all keys* k *in the range of* Gen*, and*
(2) A *and* Tgs *are in the message space of* $\mathcal{E}$*.*

Our MAC (Figure 4, Remark 3.4) is friendly w.r.t. HE schemes that support additive homomorphism over the reals, e.g., CKKS [21].

Proposition 3.1 (Our MAC is friendly). *The MAC in Figure 4 is* $\mathcal{E}$*-friendly w.r.t. any HE scheme* $\mathcal{E}$ *supporting additive homomorphism over the reals and whose message space contains the integers.*

Proof. The proof follows from Theorem 3.2, where we prove that Vrf evaluates a linear polynomial over the reals. □

The correctness of the MAC in homomorphic verification (HE-correctness) is straightforward to define; moreover, it follows immediately from the correctness of homomorphic computation. The formal details appear in Appendix C.2.

Next, we extend the definition of (one-time) unforgeability to the ciphertexts domain (*HE-unforgeability*). I.e., we define unforgeability in settings where the adversary submits message and tag in *encrypted form*. The unforgeability requirement is that homomorphic verification decrypts to reject, whenever the messages and tag are rejected in cleartext form.

Definition 3.6 (HE-unforgeability). *Let* $\mathcal{Q} = (\mathsf{MAC.Gen}, \mathsf{Mac}, \mathsf{Vrf})$ *be a MAC scheme for message space* A*, and* $\mathcal{E} = (HE.\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *a* C*-homomorphic public-key encryption scheme. We say that* $\mathcal{Q}$ *is* (one-time) $\mathcal{E}$-unforgeable *if for every ppt adversary* $\mathcal{A}$*,*

$$\Pr[\textit{Mac-Forge}^{\textit{one-time}}_{\mathcal{A},\mathcal{Q},\mathcal{E}}(\lambda) = 1] \leq \mathsf{neg}(\lambda)$$

where $\textit{Mac-Forge}^{\textit{one-time}}_{\mathcal{A},\mathcal{Q},\mathcal{E}}$ *is the following extension of the experiment described in Definition B.1:*

The $\textit{Mac-Forge}^{\textit{one-time}}_{\mathcal{A},\mathcal{Q},\mathcal{E}}(\lambda)$ *experiment.*

(1) *Keys* $\mathsf{k} \leftarrow \mathsf{MAC.Gen}(1^{\lambda})$ *and* $(pk, sk) \leftarrow HE.\mathsf{Gen}(1^{\lambda})$ *are generated, and* pk *are given to* $\mathcal{A}$*.*
(2) $\mathcal{A}$ *chooses a message* $\mathsf{msg}' \in \mathsf{A}$*, and is given* $\mathsf{tag}' \leftarrow \mathsf{Mac}_{\mathsf{k}}(\mathsf{msg}')$*.*
(3) $\mathcal{A}$ *outputs a pair ciphertexts* $(c_{\mathsf{msg}}, c_{\mathsf{tag}})$*.*
(4) *The experiment's output is* 1 *if-and-only-if:*
   - $\mathsf{Dec}_{sk}(c_{\mathsf{msg}}) \neq \mathsf{msg}'$*, and*
   - $\mathsf{Dec}_{sk}\left(\mathsf{Eval}_{pk}(\mathsf{Vrf}_{\mathsf{k}}; c_{\mathsf{msg}}, c_{\mathsf{tag}})\right) = \mathsf{acc}$*.*

Two remarks. First, the above definition easily extends to adversaries receiving arbitrarily many $(\mathsf{msg}', \mathsf{tag}')$ pairs (rather than a single pair) in the standard way; in this case we denote the experiment by $\mathsf{Mac\text{-}Forge}_{\mathcal{A},\mathcal{Q},\mathcal{E}}(\lambda)$. Second, the definition can easily be adapted to concrete security settings, where the MAC initialize with concrete (public) parameters pp rather than a security parameter. We call such a scheme ρ*-one-time HE-unforgeable*, if for every ppt adversary A, $\Pr[\mathsf{Mac\text{-}Forge}^{\text{one-time}}_{\mathcal{A},\mathcal{Q},\mathcal{E}} = 1] \leq \rho$ (cf. Remark B.2).

We show that unforgability in cleartext domain does *not* imply unforgability in ciphertext domain. We remark that although we focus here on one-time MAC, the attack easily extends to adversaries receiving arbitrarily many valid $(\mathsf{msg}', \mathsf{tag}')$ pairs.

Proposition 3.2 (Unforgability $\not\Longrightarrow$ HE-unforgeability). *For every CPA-secure HE scheme* $\mathcal{E}$*, and every (*$\mathcal{E}$*-friendly, one-time) unforgeable MAC* $\mathcal{Q}$*, there exists a CPA-secure HE scheme* $\widehat{\mathcal{E}}$ *and a ppt adversary* $\mathcal{A}$ *s.t.:*

$$\Pr[\textit{Mac-Forge}^{\textit{one-time}}_{\mathcal{A},\mathcal{Q},\widehat{\mathcal{E}}}(\lambda) = 1] \geq 1 - \mathsf{neg}(\lambda)$$

Proof. The proof appears in Appendix C.2. □

In contrast, we prove that unforgeability implies HE-unforgeability if the HE scheme is fully composable w.r.t. {Vrf}. Looking ahead, this result is essential for the security of our protocols.

Proposition 3.3 (Unforgability $\Longrightarrow$ HE-unforgeability, if the HE is fully composable). *For every HE scheme* $\mathcal{E}$ *and* $\mathcal{E}$*-friendly MAC scheme* $\mathcal{Q} = (\mathsf{MAC.Gen}, \mathsf{Mac}, \mathsf{Vrf})$*, if* $\mathcal{E}$ *is fully composable w.r.t.* {Vrf}*, then unforgeablility of* $\mathcal{Q}$ *implies that it is HE-unforgeable. Moreover, security does not deteriorate: if* $\mathcal{Q}$ *is* ρ*-unforgeablility then it is* ρ*-HE-unforgeable.*

Proof. The proof appears in Appendix C.2. □

HE-correctness and HE-unforgeability w.r.t. approximate HE. In approximate HE schemes, like CKKS, the homomorphic computation preserves only the most significant digits, while corrupting the lower digits with noise. To address this we scale up the input, to ensure that the data content throughout the homomorphic computation resides in the preserved digits; upon decryption, we round the output to the preserved digits, and scale back down. This

ensures that HE-correctness and HE-unforgeability hold also in approximate HE schemes, like in exact ones.

4 Applications of our MAC

We utilize our MAC as a central tool for authenticating HE-encrypted values received from a (possibly compromised) data keeper, who is authorized to modify ciphertexts (but not the data). We present two protocols, denoted oneSidedPriv and twoSidedPriv. Both protocols guarantee privacy against the data consumer, and authenticity against the data keeper; they differ w.r.t. the privacy guarantee against the data keeper:

- oneSidedPriv: does not guarantee privacy against that data keeper (cf. Section 1, Result 2).
- twoSidedPriv: guarantees privacy against the data keeper. This protocol is an integration of our MAC with the protocol of [8].

Both protocols are comprised of three phases: a trusted setup, store and retrieve.

The entities in our protocols are denoted as follows: *data producers* are denoted dataProd, the *data keeper* is denoted dataKeeper, the data consumer is denoted dataConsmr. In [8] these entities are denoted *data producers*, *auxiliary service* and *computing server* respectively.

As an optimization we employ (two, independent) PRFs (with security against all ppt adversaries) to replace the uniform generation of the 1st share in Figure 3 and the MAC key in Figure 4. In the following we first discuss the system architecture within which our protocols are intended to be executed, and then provide the details of the protocols and their properties.

System architecture and protocol flow. Similarly to [8], our protocol is intended to be executed within a larger system, including: applications requesting the computation over encrypted data and a persistent storage resource. The keys used in the protocols are generated by a trusted setup during a preliminary execution phase, and are then distributed to the relevant entities (as specified in Figures 5 and 6).

The store protocol is initiated by a data producer dataProd with input $(index, x)$, and terminates with authenticated data uploaded to storage (in twoSidedPriv: authenticated and encrypted).

The retrieve protocol (cf. Figure 7) is initiated by a request from an application specifying the data location *index* to compute on and the requested computation. Both dataKeeper and dataConsmr hold the HE public key associated with the application; in addition, dataConsmr holds the PRF key (keys, in twoSidedPriv) associated with the data at *index*. The retrieve protocol terminates with dataConsmr holding HE ciphertexts for the requested data as well as one extra ciphertexts c_σ that decrypts to acc if-and-only-if all that data verifies (rej otherwise). Subsequently, dataConsmr executes the requested homomorphic computation on the retrieved ciphertexts for the data, and sends to the application the outcome ciphertext c_{res} together with c_σ. The application accepts c_{res} only if c_σ decrypts to acc.

For simplicity, we present the protocols w.r.t a single *index* and HE key. More generally, the application can specify a vector of indices together with a batching pattern of which data items should be packed in each ciphertext for further homomorphic computation in a single-instruction-multiple-data (SIMD) fashion. Moreover, the application can specify the HE key identifier indicating under which public key the retrieval and computation should be executed.

Description of the oneSidedPriv protocol (cf. Figure 5). The trusted setup phase entails sampling HE keys (pk, sk) and a PRF key; providing pk to dataConsmr and dataKeeper; and the PRF key to dataProd and dataConsmr (the secret key sk is only accessible to authorized applications). In store, dataProd derives a MAC key $\mathsf{k} \leftarrow PRF(index)$ and tag $\leftarrow \mathsf{Mac}_\mathsf{k}(x)$, and uploads $(index, x, \text{tag})$ to storage. In retrieve, dataKeeper fetches $(index, x, \text{tag})$ from storage, HE-encrypts x and tag, and sends the ciphertexts c_x and c_{tag} to dataConsmr; dataConsmr performs homomorphic verification over encrypted data and tag to obtain the encrypted authenticity indicator $c_\sigma \leftarrow \mathsf{Eval}_{pk}(\mathsf{Vrf}; c_x, c_{\text{tag}})$. If the HE is not fully composable w.r.t Vrf, dataConsmr bootstraps c_x, c_{tag} prior to Eval.

Description of the twoSidedPriv protocol (cf. Figure 6). The protocol is similar, but more involved.

The trusted setup entails sampling the HE keys (pk, sk) and two independent PRFs, denoted f, g; providing pk to dataConsmr and dataKeeper; and the two PRFs to dataProd and dataConsmr (the secret key sk is only accessible to authorized applications).

In store, dataProd encrypts the data using the symmetric cipher based on secret sharing specified below; computes an authenticity tag for that ciphertext; and uploads the ciphertext and tag to storage. The symmetric cipher is a HE-friendly 2-out-of-2 secret sharing scheme with a pseudo-random 1st share $s_1 \leftarrow f(index)$ (the symmetric key), and a 2nd share $s_2 \leftarrow \mathsf{Shr}_{s_1}(x)$ derived from the data and the 1st share (the symmetric ciphertext). The mac key and tag are $\mathsf{k} \leftarrow g(index)$ and tag $\leftarrow \mathsf{Mac}_\mathsf{k}(s_2)$, respectively; uploading to storage $(index, s_2, \text{tag})$.

In retrieve, dataKeeper fetches $(index, s_2, \text{tag})$ from storage, HE-encrypts s_2 and tag, and sends the ciphertexts c_2, c_{tag} to dataConsmr. dataConsmr derives the keys s_1 and k using the PRFs; performs homomorphic verification $c_\sigma \leftarrow \mathsf{Eval}_{pk}(\mathsf{Vrf}; c_2, c_{\text{tag}})$ and homomorphic reconstruction $c_x \leftarrow \mathsf{Eval}_{pk}(\mathsf{Rec}_{s_1}; c_2)$. If the HE is not fully composable w.r.t. {Vrf, Rec}, dataConsmr bootstraps c_2, c_{tag} prior to Eval.

Threat model and security guarantee. The threat model consists of honest dataProd; dataKeeper that may be compromised by a ppt adversary mounting any attack strategy (aka, malicious); dataConsmr that may be compromised by a ppt adversary that follows the protocol specification (aka, semi-honest). The adversary cannot control both dataKeeper and dataConsmr simultaneously (two-server model). Capturing malicious dataConsmr is discussed in Remark D.4.

Theorem 4.1. *The oneSidedPriv protocol (cf. Figure 5), for every input* $(index, x) \in \{0,1\}^* \times \mathsf{A}$ *and the corresponding output* (c_x, c_σ)*, satisfies the following w.r.t. the keys* (pk, sk) *and z generated in its trusted setup phase:*

ppt & Correct. dataConsmr, dataKeeper *and the party executing the trusted setup are all ppt, and*

$$\Pr[\mathsf{Dec}_{sk}(c_x, c_\sigma) = (x, \text{acc})] \geq 1 - \text{neg}(\lambda)$$

Private. *For every ppt* dataConsmr**, ppt distinguisher* $\mathcal{D}$ *who chooses index and* $x_0, x_1 \in \mathsf{A}$ *s.t.* $|x_0| = |x_1|$*, and denoting*

by $\mathsf{view}(x_i)$ *the input, randomness and messages received by* $\mathsf{dataConsmr}^*$ *in the protocol, the following holds:*

$$\left|\Pr[\mathcal{D}(\mathsf{view}(x_0)=1]-\Pr[\mathcal{D}(\mathsf{view}(x_1)=1]\right|\leq \mathsf{neg}(\lambda)$$

Sound. *For every ppt* $\mathsf{dataKeeper}^*$ *and every* $x'\neq x$,

$$\Pr[\mathsf{Dec}_{sk}(c_x,c_\sigma)=(x',\mathsf{acc})]\leq \mathsf{neg}(\lambda)$$

Theorem 4.2. *The twoSidedPriv protocol (cf. Figure 6 is* ***ppt****,* ***correct*** *and* ***sound*** *as specified in Theorem 4.1, as well as:*

Private. *For every ppt* $P^*\in\{\mathsf{dataConsmr}^*,\mathsf{dataKeeper}^*\}$*, ppt distinguisher* $\mathcal{D}$ *who chooses index and* $x_0,x_1\in\mathsf{A}$ *s.t.* $|x_0|=|x_1|$*, and denoting by* $\mathsf{view}_{P^*}(x_i)$ *the input, randomness and messages received by* P^* *in the protocol, the following holds:*

$$\left|\Pr[\mathcal{D}(\mathsf{view}_{P^*}(x_0)=1]-\Pr[\mathcal{D}(\mathsf{view}_{P^*}(x_1)=1]\right|\leq \mathsf{neg}(\lambda)$$

Remark 4.3 (Optimization: a single authenticity indicator in retrieval of multiple data items). *For simplicity, we have presented our protocols w.r.t. retrieving a HE-ciphertext for a single data item. More generally,* dataConsmr *can retrieve HE-ciphertexts for multiple data items stored by* dataProd*. In this case, to minimize the work load of the entity obtaining the (encrypted) authenticity indicator, our protocol compresses all authenticity indicators* $c_{\sigma_1},c_{\sigma_2},\ldots$ *to a single indicator:* $c_\sigma=\sum_i c_{\sigma_i}^2$*. We note that* $\mathsf{Dec}_{sk}(c_\sigma)=0$ *(interpreted as* acc*) if-and-only-if* $\mathsf{Dec}_{sk}(c_{\sigma_i})=0$ *for all* i*.*

Remark 4.4 (Generic protocols). *The protocol in Figure 5 (respectively, Fig. 6) is generic: it can be instantiated with any HE and MAC (resp., HE, MAC and secret sharing) satisfying the requirement specified there. The protocols can be instantiated, for example, with:*

- *Any HE that supports additive homomorphism over the reals and is fully composable w.r.t* $\{\mathsf{Vrf}\}$ *(resp.,* $\{Rec,\mathsf{Vrf}\}$*) together with our MAC specified in Figure 4 (resp. our MAC and secret sharing scheme of Figure 3).*
- *Any HE that supports additive homomorphism over* $\mathbb{Z}_p^n$ *and is fully composable w.r.t* $\{\mathsf{Vrf}\}$ *(resp.,* $\{Rec,\mathsf{Vrf}\}$*), together with the classic MAC for* $\mathbb{Z}_p^n$ *specified in Figure 2 (resp. together with that MAC and the classic 2-out-of-2 additive secret sharing scheme* (Shr,Rec) *for* $\mathbb{Z}_p^n$ *defined by* $(s_1,s_2)\leftarrow\mathsf{Shr}(x)$ *for uniformly random* $\mathbf{s_1}\in\mathbb{Z}_p^n$ *and* $\mathbf{s_2}:=\mathbf{x}-\mathbf{s_1} \bmod p$*, and* $\mathbf{s_1}+\mathbf{s_2} \bmod p\leftarrow Rec(\mathbf{s_1},\mathbf{s_2})$*).*

Common parameters: HE $\mathcal{E}=(\mathsf{HE.Gen},\mathsf{Enc},\mathsf{Dec},\mathsf{Eval})$ and MAC $\mathcal{Q}=(\mathsf{MAC.Gen},\mathsf{Mac},\mathsf{Vrf})$ s.t. $\mathcal{Q}$ is $\mathcal{E}$-friendly and $\mathcal{E}$ is fully composable w.r.t Vrf, C-homomorphic, and CPA-secure. A PRF $\{f_z\colon\{0,1\}^*\to\mathcal{K}\}_{z\in\{0,1\}^\lambda}$.

Parties: dataProd, dataConsmr and dataKeeper.

Trusted setup: Sample independent keys $(pk,sk)\leftarrow\mathsf{HE.Gen}(1^\lambda)$ and $z\leftarrow_R\{0,1\}^\lambda$. Give pk to dataConsmr and dataKeeper; and f_z to dataProd and dataConsmr.

Storage: store is executed by dataProd on input $(index,x)\in\{0,1\}^*\times\mathsf{A}$, as follows.

(1) Compute $\mathsf{k}\leftarrow f_z(index)$
(2) Compute $\mathsf{tag}\leftarrow\mathsf{Mac}_\mathsf{k}(x)$
(3) Upload to storage $(index,x,\mathsf{tag})$ with access authorized to dataKeeper

Retrieval: retrieve is executed by dataConsmr and dataKeeper, where dataConsmr has input *index* (dataKeeper has no input), as follows.

(1) dataConsmr sends *index* to dataKeeper;
(2) dataKeeper does the following:
  (a) Downloads from storage $(index,x,\mathsf{tag})$,
  (b) Compute $c_x\leftarrow\mathsf{Enc}_{pk}(x)$ and $c_\mathsf{tag}\leftarrow\mathsf{Enc}_{pk}(\mathsf{tag})$
  (c) Send (c_x,c_tag) to dataConsmr;
(3) dataConsmr computes $\mathsf{k}\leftarrow f_z(index)$ and $c_\sigma\leftarrow\mathsf{Eval}_{pk}(\mathsf{Vrf}_\mathsf{k};c_x,c_\mathsf{tag})$, and outputs (c_x,c_σ)

Figure 5: The oneSidedPriv protocol.

5 Empirical Evaluation

We implemented the MAC (Fig. 4), secret sharing (Fig. 3) and the protocols oneSidedPriv and twoSidedPriv (cf. Section 4) into a system named authCSHER, supporting *authenticated Compact Storage with HE-Retrieval*. We mounted authCSHER on AWS EC2 computing instances using S3 buckets for storage, and ran extensive experiments to test its performance and scalability, using a comprehensive set of benchmarks for real-world scenarios. In the following we present implementation details (Section 5.1), experimental setup (Section 5.2) and results (Section 5.3).

5.1 The Implemented System

Software. For the HE scheme we use CKKS [21] implementation in Microsoft SEAL version 4.1.1 [46]. The PRF is based on HKDF with SHA-512 using Crypto++ version 8.9.0. HKDF keys are kept in cache memory of the relevant parties and are reused across executions of the protocol. The system is implemented in C++17, compiled with g++ version 11.4.0 on Ubuntu 22.04 flavored machines.

Hardware. Experiments are executed on AWS EC2 computing instances using an AWS S3 buckets storage. For each of the entities dataProd, dataKeeper, dataConsmr we allocated a separate machine, all under the same AWS subnet. The dataKeeper and dataConsmr communicate via TCP/IP socket. Data is stored in a single S3 bucket, in the same AWS region as the EC2 machines. The EC2 instances are of type m5.2xlarge with 8 Intel(R) Xeon(R) Platinum 8175M 2.5GHz CPUs, 32GB RAM and up to 10Gbps of network bandwidth.

Parallel processing. Our implementation for the data producer dataProd and the data keeper dataKeeper entities is single threaded. The data consumer dataConsmr uses two threads, one for receiving data from dataKeeper and one for processing it.

oneSidedPriv vs. twoSidedPriv execution modes. Our system can be executed in either oneSidedPriv of twoSidedPriv mode (cf. Section 4). In oneSidedPriv mode, dataProd uploads to storage authenticated data in cleartext. In twoSidedPriv mode, dataProd first encrypts the data (using a symmetric cipher based on the secret sharing scheme), and uploads authenticated ciphertexts; these ciphertexts are securely transciphered to HE during retrieval.

Unbatched vs. batched execution modes. Our tag generation algorithm (Mac) can be executed on a varying number of data items

Common parameters: A C-homomorphic CPA-secure HE scheme $\mathcal{E}$ = (HE.Gen, Enc, Dec, Eval) that is fully composable w.r.t the reconstruction and verification algorithms {Rec, Vrf} of the secret-sharing and MAC schemes specified next. An $\mathcal{E}$-friendly 2-out-of-2 secret sharing scheme $\mathcal{S} = (\mathsf{Shr}_{s_1}, \mathsf{Rec}_{s_1})$ for a domain A with a random 1st share $s_1 \in \mathcal{S}_1$ and a uniform 2nd share $s_2 \in \mathcal{S}_2$ (cf. Definition 2.3). An $\mathcal{E}$-friendly MAC $\mathcal{Q}$ = (MAC.Gen, Mac, Vrf) for message space $\mathcal{S}_2$. PRFs $\{f_z : \{0,1\}^* \to \mathcal{S}_1\}_{z\in\{0,1\}^\lambda}$ and $\{g_z : \{0,1\}^* \to \mathcal{K}\}_{z\in\{0,1\}^\lambda}$.

Parties: dataProd, dataConsmr and dataKeeper.

Trusted setup: Sample independent keys $(pk, sk) \leftarrow \mathsf{HE.Gen}(1^\lambda)$ and $z, z' \leftarrow_R \{0,1\}^\lambda$. Give pk to dataConsmr and dataKeeper; and f_z and $g_{z'}$ to dataProd and dataConsmr.

Storage: store is executed by dataProd on input $(index, x) \in \{0,1\}^* \times \mathsf{A}$, as follows.

(1) Compute $s_1 \leftarrow f_z(index)$ and $\mathsf{k} \leftarrow g_{z'}(index)$
(2) Compute $s_2 \leftarrow \mathsf{Shr}_{s_1}(x)$ and $\mathsf{tag} \leftarrow \mathsf{Mac}_\mathsf{k}(s_2)$
(3) Upload to storage $(index, s_2, \mathsf{tag})$ with access authorized to dataKeeper

Retrieval: retrieve is executed by dataConsmr and dataKeeper, where dataConsmr has input *index* (dataKeeper has no input):

(1) dataConsmr computes $s_1 \leftarrow f_z(index)$, $\mathsf{k} \leftarrow g_{z'}(index)$, and sends *index* to dataKeeper
(2) dataKeeper does the following:
  (a) Download from storage $(index, s_2, \mathsf{tag})$
  (b) Compute $c_2 \leftarrow \mathsf{Enc}_{pk}(s_2)$, $c_\mathsf{tag} \leftarrow \mathsf{Enc}_{pk}(\mathsf{tag})$
  (c) Send (c_2, c_tag) to dataConsmr
(3) dataConsmr computes $c \leftarrow \mathsf{Eval}_{pk}(\mathsf{Rec}_{s_1}; c_2)$ and $c_\sigma \leftarrow \mathsf{Eval}_{pk}(\mathsf{Vrf}_\mathsf{k}; c_2, c_\mathsf{tag})$, and output (c, c_σ)

Figure 6: The twoSidedPriv protocol.

n (for n the parameter used in our MAC, Fig. 4). When the tag is computed for the data items one-by-one, i.e., setting $n = 1$, we call the execution *unbatched*. When a tag is computed for each batch of $n > 1$ data items, we call the execution *batched*. We note that in unbatched mode, retrieve includes the optimization described in Remark 4.3 for authenticity indicator aggregation.

Parameters. We instantiate our system with the following parameters for our MAC and for the CKKS HE. Our MAC is set to at least 64-bits information-theoretic security (following the recommendations for universal hashing based MAC from RFC 4418 [36]), using amplification (cf. Remark 3.4): in unbatched mode, we employ 18-bit prime p and perform 4 independent repetition of our MAC, whereas in batched mode, we employ 12-bits prime p and perform 6 MAC repetitions, achieving in both cases 72-bits statistical security. The values for p were set as to provide sufficiently high precision in all our experiments. We note that the choice of p does not limit the input space, as inputs can be represented in base-p for any $p \geq 2$. The CKKS parameters are set to 128-bits security parameter, cyclotomic polynomial of degree 32768, with 60-bits of precision for the plaintext moduli at each level of homomorphic computation with $(60, 60, 60, 60)$ in batched mode, and $(60, 60, 60, 60, 60)$ in unbatched mode). We note that we set the precision to be sufficiently large so that the noise introduced by CKKS does not influence the integral part of the computed value. The number of slots for data items that can be packed in each ciphertext, denoted slots, is half the degree of the cyclotomic polynomial.

Optimizations and complexity of our system. The implementation of our system performs storage and communication optimizations as described in Appendix D.2.

Treatment of non fully-composable HE. Our experimental results executed using the SEAL HE library are under the heuristic assumption that CKKS is fully-composable w.r.t. {Vrf} in oneSidedPriv protocol ({Vrf, Rec} in twoSidedPriv). In the following we denote this by Composability heuristic.

In case Composability does not hold, we bootstrap the ciphertexts given as input to $\mathsf{Eval}(\mathsf{Vrf}; \cdot, \cdot)$ (resp., $\mathsf{Eval}(\mathsf{Rec}; \cdot, \cdot)$) prior to their homomorphic evaluation; this guarantees full composability w.r.t Vrf (resp., Rec, Vrf) by [41].

For bootstrapping we utilize the openFHE library [11] CKKS implementation (because SEAL does not currently support bootstrapping for CKKS). We implemented, in openFHE, our oneSidedPriv pipeline entailing encrypting with CKKS the (message,tag) pair uploaded by dataProd, bootstrapping the ciphertexts, and then performing homomorphic tag verification. with 2^{17} cyclotomic polynomial and 64K slots available. We executed this implementation in AWS c5.18xlarge instance. The runtime to encode, encrypt and bootstrap one ciphertext with 2^{16} data slots, was under 38.33s (amortized, per data item, runtime of 0.58ms).

5.2 The Experimental Setup

Data. We ran our system for storing and HE-retrieving up to 2M data items. Our experiments are executed with four different numbers of data items: 16384, 98304, 507904 and 2031616. Each data item is synthetically generated, from the domain $[0..p-1]$ (for p the parameter with which we execute our MAC, cf. Figure 4), represented as double-precision numbers according to the IEEE 754 standard [2].

Experiments executing our system. We executed our system in both oneSidedPriv and twoSidedPriv modes; and in both unbatched and batched modes. Each experiment is repeated 10 times, taking the average. We ran both end-to-end and microbenchmarks experiments. We measured storage size, runtime and communication.

EtM baseline experiments. For comparison purposes we implemented the EtM solution, where dataProd uploads (compressed) authenticated HE ciphertexts, and dataConsmr verifies ciphertexts authenticity at retrieval. We use HMAC with SHA-256 for authentication, and Zstandard library for compression. We executed this EtM baseline in the same AWS with S3 bucket computing environment as our system; employing the same CKKS Microsoft SEAL implementation for the HE scheme, as in our system, and with the same parameters. Performance is evaluated in both unbatched and batched modes, corresponding to the unbatched and batched experiments on our system. In unbatched mode, we place a single data item in each HMACed ciphertext; in batched mode, slots data items are packed in each HMACed ciphertext.

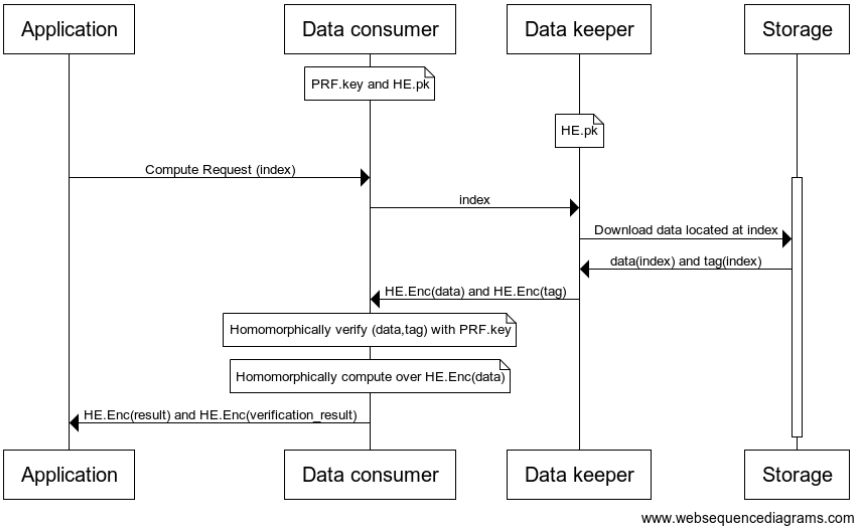

Figure 7: System architecture for oneSidedPriv retrieve protocol

Comparing our system to other solutions. We compare our system against the four following alternative solutions: (1) [20]'s data authenticity proof component, which offers the same authenticity and privacy guarantee as when executing our system in oneSidedPriv mode. (2) [8]'s compact storage with HE-retrieval system, which offers the same privacy guarantee as when executing our system in twoSidedPriv mode, *but provides no authenticity* guarantee. (3) The Encrypt-then-MAC (EtM) approach of storing and downloading authenticated data directly in HE-encrypted form. (4) [3]'s authenticated transciphering from AES-GCM to CKKS, which offers authenticity and privacy guarantee as when executing our system in twoSidedPriv mode (see Table 5 and Paragraph 5.3). For the EtM, we executed experiments to measure performance. For the other three we report figures from their publications [3, 8, 20].

5.3 Results

Performance of our MAC (cf. Section 1, Result 1). The performance of our MAC scheme is as follows, when reporting amortized performance, per data item (derived from an execution over 2,031,616 data items). Key generation plus tag generation takes $1.17\mu s$ in batched mode ($10.03\mu s$ in unbatched mode). Key generation plus homomorphic verification takes, under Composability heuristic, $7.37\mu s$ in batched mode ($20.11\mu s$ in unbatched mode). Otherwise, verification takes $0.33ms$ and $4.68ms$ in batch and unbatched mode, respectively. We note that key generation, in the unbatched mode, was conducted item-by-item using HMAC – which takes the bulk of the runtime; in batched mode, key generation is computed in bulk, using HKDF, which is considerably faster. The tag size is smaller than 1 byte in batched mode (32 bytes in unbatched mode).

Performance of our system in oneSidedPriv mode and comparison to [20] (cf. Section 1, Result 2). The generation of our authenticity proof for encrypted data consists of encrypting, with HE, the (cleartext) tag that was uploaded by dataProd. Verification consists of homomorphic tag verification. We report amortized performance, per data item, derived from an execution over 2,031,616 data items. Under Composability heuristic, proof generation time, verification time and proof size are $0.21\mu s$, $7.36\mu s$ and 13.94 bytes, respectively, in batched mode ($14.01\mu s$, $20.10\mu s$ and 1024 bytes, respectively, in unbatched mode). In case Composability does not hold, proof size and generation time are unchanged; whereas verification time becomes $0.92ms$ and $5.26ms$ in batch and unbatched mode, respectively.

We compare the performance of our authenticity proof to [20] (where both ours and [20] are in batched mode). Performance are those reported in their usecase that is closest to ours in terms of the encryption parameters (cf. *location based activity tracking* there). We note that the performance of [20] reported here, incorporates all their proposed optimizations, including their *batching optimizations* and *Random Integrity Checks*; the latter entails verifying authenticity only on a random subset of data points, which protects against covert adversaries – i.e., weaker than the malicious adversary addressed in our solution. [20] provides another solution that does protect against malicious adversaries, albeit with a performance penalty (i.e., larger proof size and proof and verification times). In terms of proof generation time, we demonstrate a significant improvement over [20], with a reduction in generation time of more than four orders of magnitude; concretely, $1.37\mu s$ in ours solution vs. $6.1\times10^4\mu s$ in theirs. Verification time also follows a similar trend, with our implementations requiring $7.36\mu s$ ($0.092\cdot10^4\mu s$ if Composability does not hold), compared to $3.2\times10^4\mu s$. Our system has a significantly lower proof size requiring 13.94 bytes per data item whereas [20] requires approximately $1.65\cdot10^5$ bytes per data item. See Table 1.

Performance of our system in twoSidedPriv mode and comparison to [8], EtM and [3]. Our system's amortized performance per data item, in an execution with 2M data items in batched mode (i.e., one tag) is: 8.43 bytes storage, $1.66\mu s$ store time, and

	Polynomial Ring	Modulus log q	Proof generation time (μs)	Verify time (μs)	Proof size (bytes)
Chatel et al. [20]	2^{13}	184	6.1×10^4	3.2×10^4	1.65×10^5
Ours (Batched)	2^{15}	240	0.21	7.36	13.94

Table 1: Comparison of Our vs. [20] authenticity proof. The proofs are over batched data, with batch sizes of 16K and 2K data items in Our and [20] respectively. The table reports time and size performance amortized per data item. Our reported Verify time assumes Composability (otherwise, we bootstrap data and tag, which increases the amortized verification runtime to $\mathbf{0.09}\times 10^4 \mu s$ (the proof size and its generation time are not affected).

8.3μs retrieve time under Composability heuristic (1.5ms otherwise); cf. Table 2, Batched. In unbatched mode with 2M data items and 2M tags, our system's amortized performance per data item is: 40bytes storage, 10.91μs store time, and 28.91μs retrieve time under Composability heuristic (5.85ms otherwise); cf. Table 2, Unbatched. Microbenchmarks detailing are reported in Tables 3-4.

Comparison to [8]. Table 6 presents a detailed comparison between the *unauthenticated* solution [8, Tables 2 and 4]), called CSHER, and our *authenticated* solution (with figures derived from our Table 4), under Composability heuristic. In high level, our batched solution demonstrates superiority over CSHER in most metrics, whereas the overhead introduced by the MAC per-item is more evident in our unbatched solution. In terms of *storage size*, our batched solution offers a slight advantage over CSHER, likely due to minor implementation differences. However, our unbatched mode requires larger storage to accommodate the MAC repetitions per item necessary to achieve the target statistical security. The *store runtime* of our batched solution is 1.5×-slower than CSHER in batched mode (10×, in unbatched mode) due to the tag computation. *Retrieve times* for both our batched and unbatched solutions benefit significantly from parallel execution, as outlined in Section 5.1 and the optimizations detailed in Appendix D.2. Despite requiring MAC repetitions, our batched mode (four MAC repetitions), achieves performance comparable to CSHER, and our unbatched mode (six MAC repetitions) is within a factor of three and a half compared to CSHER. *Communication* in our batched mode, is 20% larger than CSHER (7.5× larger, in unbatched mode). Notably, the communication time in our batched mode is 33% faster than CSHER (and only 3.58× slower, in unbatched mode). This improvement is primarily due to the use of TCP sockets and parallel processing in our implementation, in contrast to the single-threaded HTTP communication utilized by CSHER. Crucially, our solution provides data integrity which is absent in CSHER.

Comparison to EtM. The comparison (cf. Table 2 and Figure 8) to the EtM baseline shows that, under Composability heuristic, our system offers a substantial improvement in storage size with a minor degradation in runtime:

- *Compact storage size:* 11× better than the baseline in batched mode, and $5.2{\cdot}10^4\times$ better in unbatched mode.
- *Fast end-to-end store-then-retrieve runtime:* only 2× slower than the baseline in batched mode, and $2.3{\cdot}10^3\times$ faster than the unbatched baseline time.

In further details, our storage runtime and size are always faster and smaller than the EtM baseline. Our retrieval runtime is faster than the baseline in batched mode, but slower in unbatched mode. Essentially, our storage is faster because we store items of double datatype (rather than encrypting and storing HE ciphertexts in the baseline); whereas our retrieval is slower because it includes HE encryption, as well as homomorphic evaluation of Vrf and Rec (rather than fetching from storage previously encrypted HE ciphertexts in the baseline).

Comparison to [3]. Table 5 presents the comparison to Aharoni et al. [3]'s transciphering from AES-GCM-128 to CKKS. The comparison shows that our system offers runtime improvement and significant monetary savings, albeit the two-server model. Concretely, [3] are running on an A100 SXM4 80GB GPU with 6912 CUDA cores hardware compared to our much cheaper and commonly used c5.18xlarge hardware (M5.2xlarge in our experiments under Composability heuristic). Their runtime per 8-bytes data item is 6.5 milliseconds (cf. 1.5ms in our experiment on c5.18xlarge that include bootstrapping to guarantee that Composability holds), i.e., 4.33× slower than our runtime with bootstrapping (7,386× slower than our runtime when executing our system under the Composability heuristic). In terms of monetary cost –the product of hardware cost times runtime– we achieve a 55× cost reduction compared to Aharoni et al. [3] by our solution that includes bootstrapping (and nearly $10^5\times$ reduction in cost by our solution executed under the Composability heuristic).

Monetary cost. We illustrate in Figure 9 the monetary implications of using our solution compared to the EtM baseline, in batched and unbatched setting respectively, by extrapolating our empirical measurements to massive amounts of data. We rely on AWS pricing [1] for M5.2xlarge EC2 machine and S3 bucket as used in our system ($0.384 per computing hour, and $0.021 for 1GB storage per month). Consider a data lake of 25PB of data. AWS storage cost for our batched solution is approximately $0.55M per month; so our solution saves roughly $5.45M per month in storage costs compared to the baseline solution of directly storing HE ciphertexts. Moreover, even if the data is stored unbatched our solution saves roughly $3.25M per month in storage costs compared to the baseline solution. Furthermore, even when accounting for the overhead in HE-retrieval compared to directly retrieving HE ciphertexts – our solution is cost effective for HE-retrieval of almost 10^{16} data items in batched mode per month if bootstrapping is not required and 10^{13} with bootstrapping. Furthermore, even for data in the unbatched mode our solution outperforms the batched baseline cost up to 10^{14} without bootstrapping and 10^{12} if bootstrapping is required. This is because, initially, the storage cost dominates the overall cost – and so, our solution offers substantial savings; but, gradually, the runtime cost becomes the dominating factor

	Batched			Unbatched		
	Storage size (bytes)	Runtime (μs)		Storage size (bytes)	Runtime (μs)	
		Store	Retrieve		Store	Retrieve
EtM Baseline	**96.01**	**3.29**	**1.48**	**2,097,514.16**	**59,205.02**	**33,300.46**
Ours	**8.43**	**1.66**	8.30	**40.00**	**10.91**	**28.91**

Table 2: Runtime time (microseconds) and storage (bytes) per data item, in execution on 2031616 data items

	Data Producer		Data Keeper			Data Consumer				Communication	
Benchmark Operations	**Share + MAC**	**Upload**	**Download**	**Encrypt**	**Serialize**	**Desrialize**	**Derive**	**Reconstruct and Verify**	**Square diff**	**Time**	**Size (bytes)**
Step #	1, 2	3	2a	2b	–	–	3	3	4.3	2c	2c

Table 3: Microbenchmarks mapping to steps in the protocol as specified in Figure 6

	Data Producer		Data Keeper			Data Consumer				Comm.	
Benchmark Operations	**Share + MAC**	**Upload**	**Download**	**Encrypt**	**Serialize Data**	**Deserialize Data**	**Derive**	**Reconstruct and Verify**	**Square diff**	**Time**	**Size (bytes)**
Ours (Batched)	1.32	0.34	0.16	3.08	0.46	0.12	1.29	7.02		0.07	206
Ours (Unbatched)	10.21	0.69	0.73	17.50	2.94	0.76	8.85	12.60	4.76	0.43	1279

Table 4: Microbenchmarks. Runtime(μs), storage(bytes) and communication per data item, in execution on 2031616 data items

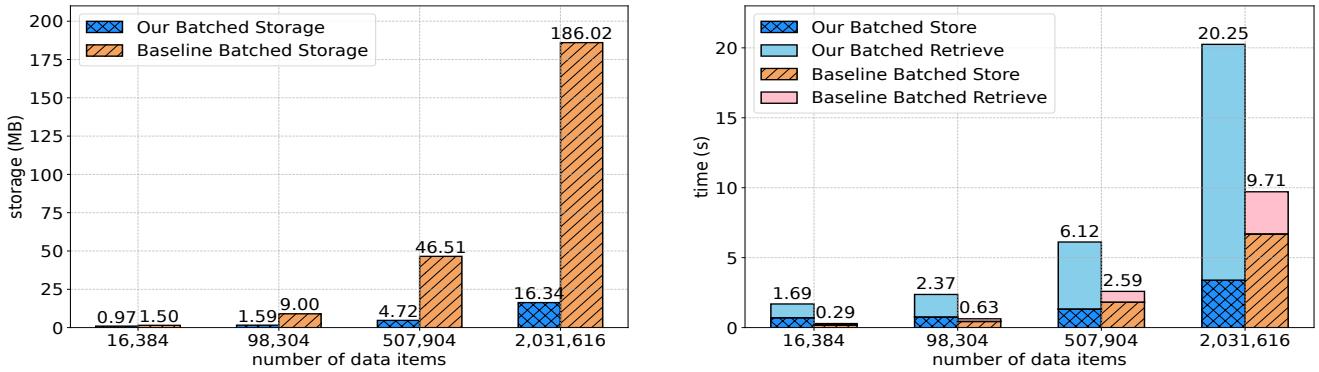

Figure 8: Storage and runtime of our solution vs. EtM baseline on varying batch sizes

	Storage (bytes)	Runtime (ms)	Hardware	Spec (Cores\Memory)	Cost (USD$\times 10^{-6}$)
Aharoni et al. [3]	8.000	6.500	GPU	6912 CUDA\80GB	73.830
Ours (Batched)	8.430	0.001	CPU	8 vCPUs\32 GB	0.001
Ours + bootstrap (Batched)	8.430	1.500	CPU	72 vCPUs\144 GB	1.500

Table 5: Comparison of Ours vs. Aharoni et al. [3]. Performance is amortized per 8-byte data item. [3] transcipher from AES-GCM-128 to CKKS with batched authentication. Our reported runtime include bootstrapping (under Composability heuristic our runtime is 0.0083ms).

(as storage is constant in our experiment, whereas the runtime is linearly increasing in the number of retrieved data items), leading to a linear cost overhead in our solution.

HE-Retrieve then HE-Eval Decision Tree. As an application example, we applied homomorphic evaluation of a decision trees prediction on the retrieved ciphertexts. That is, dataConsmr homomorphically verified the authenticity of the retrieved secret

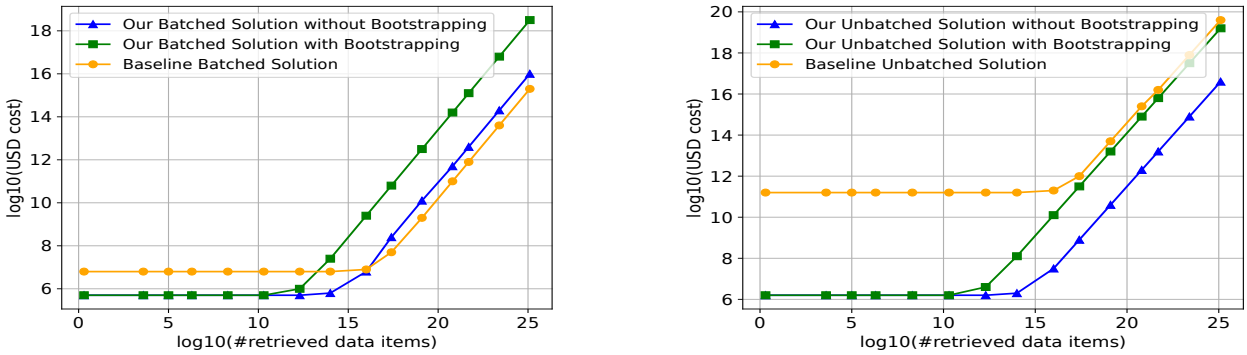

Figure 9: AWS cost (USD) for storing 25PB and retrieving up to 10^{25} data items, in log-log scale.

	Storage Size (bytes)	Runtime (μs)		Communication		Integrity
		Store	Retrieve	Time (μs)	Size (bytes)	
Akavia et al. [8]	8.25	0.98	8.76	0.12	**170.83**	×
Ours (Batched)	**8.43**	**1.66**	**8.30**	**0.08**	206.04	✓
Ours (Unbatched)	40.00	10.91	28.91	0.43	1279.77	

Table 6: Comparison of Ours vs. Akavia et al. [8]. Figures are amortized per data item, in execution on 2031616 data items.

shared, homomorphically reconstructed the data from them, and then homomorphically evaluated a decision tree model on that data. We utilized for the latter the homomorphic decision tree evaluation algorithm from [7], using the open source code from [25], executed with their degree 8 polynomial approximating the step function (aka, soft-if, their), on (full) decision trees of depth 3 and 4. The tree is evaluating on 16,384 data samples, with 46 features each; for this purpose we HE-retrieved 753,664 data items (secret shares to homomorphically verified and reconstructed), packed in 46 ciphertexts (a ciphertext for each feature). The accuracy in the homomorphic computation is similar to that of computing over cleartext (less than 10^{-6} difference in the produced class likelihood scores). The amortized runtime (under Composability heuristic) for the homomorphic tree evaluation on all samples is 10.55s, i.e., 0.64ms amortized runtime per data sample.

Treatment of HE that do not satisfy Composability. In our results that avoid Composability heuristic, we bootstrap the ciphertexts given as input to $\mathsf{Eval}(\mathsf{Vrf};\cdot,\cdot)$ (resp., $\mathsf{Eval}(\mathsf{Rec};\cdot,\cdot)$) prior to their homomorphic evaluation. This guarantees full composability w.r.t Vrf (resp., Rec, Vrf) (by [41]). Our experiments on OpenFHE library show that the process of encryption, bootstrapping, and homomorphic verification takes under 38.33s per ciphertext. Moreover, this pipeline achieves the required accuracy by our verification algorithm, as well as by applications such as the homomorphic decision tree evaluation.

We note that although bootstrapping adds a substantial overhead to the verification runtime, nevertheless, our HE-retrieval time is still minor compared to common HE-computation runtimes. For example, homomorphic training of a logistic regression model [42] on a mortality dataset from [27] (containing 46k examples with 2-features) takes 25.5hrs; retrieving the data using our HE-retrieval (with bootstrapping) adds to the former less than 1% overhead (concretely, the overhead is 0.47% and 0.78% in batched and unbatched modes, respectively). Likewise, homomorphic training of decision tree models [7], and homomorphic features selection [4] have runtime in the scale of hours; so, our HE-retrieval runtime is comparatively a minor part of the total retrieve-then-evaluation computation.

6 Conclusions

We showed how to modify the classical affine MAC to support fast homomorphic verification over the reals, with application for guaranteeing data authenticity when retrieving HE ciphertexts from a, possibly faulty or compromised, data keeper who is authorized to generate and modify ciphertexts, but not their content. We then showed how to enhance the authenticity guarantee with the guarantee for compact storage and secrecy against the data keeper (on top of the data consumers) via integrating our solution with [8]. We implemented our solution into a system name authCSHER and empirically evaluated it on AWS EC2 instances and S3 storage, showing it attains nearly zero overhead over storing unauthenticated cleartext data, and fast amortized end-to-end running time (microseconds if full composability holds w.r.t. our computation, milliseconds otherwise). This provides a practical and scalable approach to authenticated, secure and efficient retrieval of HE-ciphertexts, making it well-suited for real-world applications.

7 Acknowledgment

This work was supported in part by the Center for Cyber Law & Policy at the University of Haifa in conjunction with the Israel

National Cyber Directorate in the Prime Minister's Office. We are deeply grateful to the anonymous reviewers and the revision editor for their valuable feedback, which helped improve the clarity and presentation of our contributions.

References

[1] AWS EC2 pricing. https://aws.amazon.com/ec2/pricing/on-demand/. Accessed: 2024-31-05.

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

[3] Ehud Aharoni, Nir Drucker, Gilad Ezov, Eyal Kushnir, Hayim Shaul, and Omri Soceanu. Poster: Efficient aes-gcm decryption under homomorphic encryption. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 3567–3569, 2023.

[4] Adi Akavia, Ben Galili, Hayim Shaul, Mor Weiss, and Zohar Yakhini. Privacy preserving feature selection for sparse linear regression. *Proc. Priv. Enhancing Technol.*, 2024(1):300–313, 2024.

[5] Adi Akavia, Craig Gentry, Shai Halevi, and Margarita Vald. Achievable CCA2 relaxation for homomorphic encryption. In Eike Kiltz and Vinod Vaikuntanathan, editors, *Theory of Cryptography - 20th International Conference, TCC 2022, Chicago, IL, USA, November 7-10, 2022, Proceedings, Part II*, volume 13748 of *Lecture Notes in Computer Science*, pages 70–99. Springer, 2022.

[6] Adi Akavia, Meir Goldenberg, Neta Oren, and Margarita Vald. Authenticated HE retrieval. https://github.com/ASEC-lab/auth-HE-retrieval, 2025. [Online; accessed 14-Jun-2025].

[7] Adi Akavia, Max Leibovich, Yehezkel S. Resheff, Roey Ron, Moni Shahar, and Margarita Vald. Privacy-preserving decision trees training and prediction. *ACM Trans. Priv. Secur.*, 25(3), may 2022.

[8] Adi Akavia, Neta Oren, Boaz Sapir, and Margarita Vald. CSHER: A system for compact storage with HE-Retrieval. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 4751–4768, Anaheim, CA, August 2023. USENIX Association.

[9] Martin R Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for mpc and fhe. In *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 34*, pages 430–454. Springer, 2015.

[10] Tomer Ashur, Mohammad Mahzoun, and Dilara Toprakhisar. Chaghri-a fhe-friendly block cipher. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 139–150, 2022.

[11] Ahmad Al Badawi, Andreea Alexandru, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Carlo Pascoe, Yuriy Polyakov, Ian Quah, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. OpenFHE: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915, 2022. https://eprint.iacr.org/2022/915.

[12] Carsten Baum, Ivan Damgård, Vadim Lyubashevsky, Sabine Oechsner, and Chris Peikert. More efficient commitments from structured lattice assumptions. In *International Conference on Security and Cryptography for Networks*, pages 368–385. Springer, 2018.

[13] Adda-Akram Bendoukha, Aymen Boudguiga, and Renaud Sirdey. Revisiting stream-cipher-based homomorphic transciphering in the tfhe era. In *International Symposium on Foundations and Practice of Security*, pages 19–33. Springer, 2021.

[14] Adda-Akram Bendoukha, Pierre-Emmanuel Clet, Aymen Boudguiga, and Renaud Sirdey. Optimized stream-cipher-based transciphering by means of functional-bootstrapping. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 91–109. Springer, 2023.

[15] Adda Akram Bendoukha, Oana Stan, Renaud Sirdey, Nicolas Quero, and Luciano Freitas. Practical homomorphic evaluation of block-cipher-based hash functions with applications. In *International Symposium on Foundations and Practice of Security*, pages 88–103. Springer, 2022.

[16] John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, and Phillip Rogaway. Umac: Fast and secure message authentication. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, page 216–233, Berlin, Heidelberg, 1999. Springer-Verlag.

[17] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 868–886, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[18] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.

[19] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *Proceedings of the 2017 acm sigsac conference on computer and communications security*, pages 1825–1842, 2017.

[20] Sylvain Chatel, Apostolos Pyrgelis, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. Privacy and integrity preserving computations with {CRISP}. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2111–2128, 2021.

[21] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT*, 2017.

[22] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tfhe: Fast fully homomorphic encryption over the torus. *J. Cryptol.*, 33(1):34–91, jan 2020.

[23] Carlos Cid, John Petter Indrøy, and Håvard Raddum. Fasta–a stream cipher for fast fhe evaluation. In *Cryptographers' Track at the RSA Conference*, pages 451–483. Springer, 2022.

[24] Orel Cosseron, Clément Hoffmann, Pierrick Méaux, and François-Xavier Standaert. Towards case-optimized hybrid homomorphic encryption: Featuring the elisabeth stream cipher. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 32–67. Springer, 2022.

[25] Decision trees training and prediction over encrypted data using fully homomorphic encryption. https://github.com/intuit/Decision-Trees-over-FHE, March 2023. Max Leibovich, Omer Sadeh, Boaz Sapir and Margarita Vald.

[26] Léo Ducas and Daniele Micciancio. Fhew: bootstrapping homomorphic encryption in less than a second. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 617–640. Springer, 2015.

[27] Danielle M. Ely, Anne K. Driscoll, and T. J. Matthews. *Infant mortality rates in rural and urban areas in the United States, 2014.* NCHS data brief ; v no. 285. U.S. Department of Health and Human Services, Centers for Disease Control and Prevention, National Center for Health Statistics, 2017., Hyattsville, MD, 2017.

[28] European Parliament and Council of the European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council, 2016.

[29] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012.

[30] Craig Gentry. *A fully homomorphic encryption scheme.* Stanford university, 2009.

[31] Oded Goldreich, Silvio Micali, and Avi 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.

[32] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. In Oded Goldreich, editor, *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 203–225. ACM, 2019.

[33] Jincheol Ha, Seongkwang Kim, Byeonghak Lee, Jooyoung Lee, and Mincheol Son. Rubato: Noisy ciphers for approximate homomorphic encryption. In *Advances in Cryptology – EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 – June 3, 2022, Proceedings, Part I*, page 581–610, Berlin, Heidelberg, 2022. Springer-Verlag.

[34] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography.* CRC press, 2020.

[35] Hugo Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is ssl?). In *Annual International Cryptology Conference*, pages 310–331. Springer, 2001.

[36] Ted Krovetz. UMAC: Message Authentication Code using Universal Hashing. RFC 4418, March 2006.

[37] Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 648–677. Springer, 2021.

[38] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Annual international cryptology conference*, pages 36–54. Springer, 2000.

[39] Silvia Mella and Ruggero Susella. On the homomorphic computation of symmetric cryptographic primitives. In *Cryptography and Coding: 14th IMA International Conference, IMACC 2013, Oxford, UK, December 17-19, 2013. Proceedings 14*, pages 28–44. Springer, 2013.

[40] Daniele Micciancio. Fully composable homomorphic encryption. Cryptology ePrint Archive, Paper 2024/1545, 2024.

[41] Daniele Micciancio. Fully composable homomorphic encryption. *IACR Communications in Cryptology*, 2(1), 2025.

[42] Luis Montero, Jordan Frery, Celia Kherfallah, Roman Bredehoft, and Andrei Stoian. Machine learning training on encrypted data with TFHE. In *Proceedings of the 10th ACM International Workshop on Security and Privacy Analytics*, IWSPA '24, page 71–76, New York, NY, USA, 2024. Association for Computing Machinery.

[43] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *ACM CCSW*, 2011.

[44] Yoav Nir and Adam Langley. ChaCha20 and Poly1305 for IETF Protocols. RFC 7539, May 2015.

[45] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[46] Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL, January 2023. Microsoft Research, Redmond, WA.
[47] Mark N Wegman and J Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal of computer and system sciences*, 22(3):265–279, 1981.
[48] Benqiang Wei and Xianhui Lu. Improved homomorphic evaluation for hash function based on tfhe. *Cybersecurity*, 7(1):14, 2024.
[49] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *FOCS*. IEEE, 1986.

A Supplemental Material

In the following we provide details omitted from the paper body due to space limitations.

B Supplemental Material: Preliminaries

B.1 Message Authentication Codes

Message authentication codes (MAC) enable detecting whether an adversary modified a stored message. This is formally defined as follows.

Definition B.1 (message authentication code (MAC)). *A* message authentication code *for message-space* A*, and tag-space* Tgs *consists of three ppt algorithms* $\mathcal{Q} = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrf})$*:*

- Gen *(key generation) takes as input the security parameter* λ *and outputs a key* k *with* $|\mathsf{k}| > \lambda$*; denoted* $\mathsf{k} \leftarrow \mathsf{Gen}(1^\lambda)$*. Without loss of generality,* Gen *outputs a uniformly random key from the key space, denoted* $\mathsf{k} \leftarrow_R \mathcal{K}$*.*
- Mac *(tag generation) takes as input a key* k *and a message* $\mathsf{msg} \in \mathsf{A}$*, and outputs a* $\mathsf{tag} \in \mathsf{Tgs}$*; denoted* $\mathsf{tag} \leftarrow \mathsf{Mac}_\mathsf{k}(\mathsf{msg})$*.*
- Vrf *(verification) takes as input a key* k*, a message* x*, and a* tag *and outputs* acc *or* rej *(meaning valid and invalid, respectively); denoted* $\sigma \leftarrow \mathsf{Vrf}_\mathsf{k}(\mathsf{msg}, \mathsf{tag})$*.*

The correctness *requirement is that for every* $\lambda \in \mathbb{N}$, $\mathsf{k} \leftarrow \mathsf{Gen}(1^\lambda)$ *and* $\mathsf{msg} \in \mathsf{A}$, $\mathsf{Vrf}_\mathsf{k}(x, \mathsf{Mac}_\mathsf{k}(x)) = \mathsf{acc}$*. The scheme is* existentially unforgeable under a one-time adaptive chosen-message attack *(*one-time unforgeable*, in short) if for every ppt adversary* $\mathcal{A}$*, there is a negligible function* neg*, such that:*

$$\Pr[\textit{Mac-Forge}^{\textit{one-time}}_{\mathcal{A},\mathcal{Q}}(\lambda) = 1] \leq \mathsf{neg}$$

where $\textit{Mac-Forge}^{\textit{one-time}}_{\mathcal{A},\mathcal{Q}}$ *is as defined below. (If unforgeability holds also even against unbounded adversaries, then the scheme is called* information theoretically unforgeable*.)*

The $\textit{Mac-Forge}^{\textit{one-time}}_{\mathcal{A},\mathcal{Q}}(\lambda)$ *experiment:*

(1) *A key* $\mathsf{k} \leftarrow \mathsf{Gen}(1^\lambda)$ *is generated.*
(2) $\mathcal{A}$ *chooses a message* msg'*, and is given a tag* $\mathsf{tag}' \leftarrow \mathsf{Mac}_\mathsf{k}(\mathsf{msg}')$*.*
(3) $\mathcal{A}$ *outputs a pair* $(\mathsf{msg}, \mathsf{tag})$*.*
(4) *The experiment's output is* 1 *if-and-only-if:* $\mathsf{Vrf}_\mathsf{k}(\mathsf{msg}, \mathsf{tag}) = \mathsf{acc}$ *and* $\mathsf{msg} \neq \mathsf{msg}'$*.*

Remark B.2 (MAC with concrete security). *It sometime simplifies our presentation to discuss concrete security. I.e., to consider MAC schemes indexed by public parameters* pp*, denoted* $\mathcal{Q}^{\mathsf{pp}} := (\mathsf{Gen}^{\mathsf{pp}}, \mathsf{Mac}^{\mathsf{pp}}, \mathsf{Vrf}^{\mathsf{pp}})$*. In this case* $\mathsf{Gen}^{\mathsf{pp}}()$ *and the forging experiment receive no input, and we call the scheme* ρ-one-time unforgeable *if for every ppt adversary* $\mathcal{A}$*,* $\Pr[\textit{Mac-Forge}^{\textit{one-time}}_{\mathcal{A},\mathcal{Q}} = 1] \leq \rho$ *(where* ρ *can be a function of the parameters* pp*).*

B.2 Homomorphic Encryption

Definition B.3 (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) is a randomized algorithm that 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 C-homomorphic *for a circuit family* C *if for all* $C \in C$ *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:*

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

(the probability is over all randomness in the experiment).
The scheme is CPA-secure *(aka,* semantically secure*), if no ppt adversary* $\mathcal{A}$ *can distinguish between the encryption of two equal length messages* x_0, x_1 *of his choice.*

B.3 Pseudorandom Functions

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 [34, Definition 3.25].

B.4 Secret Sharing

We write the standard definition of a 2-out-of-2 secret sharing scheme, and set some terminology.

Definition B.4 (secret sharing). *A* 2-out-of-2 secret sharing *scheme for* A *is a pair of ppt algorithms* (Shr, 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* $x \in \mathsf{A}$*,* $Rec(Shr(x)) = x$*. The (perfect)* security *requirement is that for every* $x, x' \in \mathsf{A}$ *and* $i \in \{1, 2\}$*, the following two distributions are identical:* $\{s_i\}_{(s_1,s_2) \leftarrow Shr(x)} \equiv \{s'_i\}_{(s'_1,s'_2) \leftarrow Shr(x')}$*.*

C Supplemental Material: Section 3

C.1 Supplemental Material for Section 3.1

C.1.1 Amplifying Security. For completeness of the presentation we present in Figure 10 the standard method how to amplify security via repetition.

$\underline{\mathsf{Gen}(1^\lambda)}$.. Given the security parameter λ, output $(\mathsf{k}_i)_{i\in[k]}$, where $k := \lambda/\lceil \log_2 p \rceil$ and $\mathsf{k}_i \leftarrow \mathsf{Gen}^{n,p}()$.

$\underline{\mathsf{Mac}_{(\mathsf{k}_i)_{i\in[k]}}(x)}$.. Given a key $(\mathsf{k}_i)_{i\in[k]}$ and a message x output $(\mathsf{tag}_i)_{i\in[k]}$, where $\mathsf{tag}_i \leftarrow \mathsf{Mac}^{n,p}_{\mathsf{k}_i}(x)$ for each $i \in [k]$.

$\underline{\mathsf{Vrf}_{(\mathsf{k}_i)_{i\in[k]}}(x,(\mathsf{tag}_i)_{i\in[k]})}$.. Given a key $(\mathsf{k}_i)_{i\in[k]}$, a message x, and a tag $(\mathsf{tag}_i)_{i\in[k]}$, compute $\mathsf{tag}'_i \leftarrow \mathsf{Mac}^{n,p}_{\mathsf{k}_i}(x)$ and $\mathsf{diff}_i := \mathsf{tag}_i - \mathsf{tag}'_i$ for each $i \in [k]$, and output $(\mathsf{diff}_i)_{i\in[k]}$ to be interpreted as acc if-and-only-if it is 0^k.
An optional optimization: output $\sum_{i\in[k]} \mathsf{diff}_i^2$ to be interpreted as acc if-and-only-if it is 0.

Figure 10: Amplifying the scheme from Figure 4 to have a negligible forging probability.

C.1.2 Proof of Theorem 3.2. The formal analysis of our MAC is presented next.

Proof of Theorem 3.2, Correctness. Fix $(\mathbf{k}, \mathbf{s_1}) \leftarrow \mathsf{Gen}^{n,p}()$ and $\mathbf{x} := (x_1, \ldots, x_n) \in [0..p-1]^n$ (where $\mathbf{k} \in [0..p-1]^{n+1}$, $\mathbf{s_1} \in [0..p-1]^\ell$ for $\ell := \left\lfloor \log_p(np^2) \right\rfloor$, and $\mathbf{x} \in [0..p-1]^n$, and where entries of $\mathbf{k}$ indexed from 0, and all other vectors are indexed starting from 1). Let $\mathsf{tag} := (\mathbf{s_2}, y_\mathsf{r}) \leftarrow \mathsf{Mac}^{n,p}_{\mathbf{k},\mathbf{s_1}}(\mathbf{x})$ and $\mathsf{diff} \leftarrow \mathsf{Vrf}^{n,p}_{\mathbf{k},\mathbf{s_1}}(\mathbf{x}, \mathsf{tag})$. We show that $\mathsf{diff} = 0$, (interpreted as acc).

Let y, y' be as computed in Vrf_p; since $\mathsf{diff} = y' - y$, it suffices to show that $y = y'$. To prove the latter we show that both y and y' are equal to

$$h_\mathbf{k}(\mathbf{x}) := k_0 + \sum_i k_i x_i$$

(where the arithmetic is over the reals, i.e., with no modular reduction). The fact that $y' = h_\mathbf{k}(\mathbf{x})$ follows immediately by inspection of $\mathsf{Vrf}^{n,p}_{\mathbf{k},\mathbf{s_1}}$. The fact that $y = h_\mathbf{k}(\mathbf{x})$ is argued next. By inspection of $\mathsf{Vrf}^{n,p}_{\mathbf{k},\mathbf{s_1}}$, $y = y_\mathsf{q} p + y_\mathsf{r}$ where $y_\mathsf{q} := \mathsf{Rec}^{\ell,p}_{\mathbf{s_1}}(\mathbf{s_2})$, where $(\mathbf{s_2}, y_\mathsf{r})$ are as provided in tag, which –by inspection of $\mathsf{Mac}^{n,p}_{\mathbf{k},\mathbf{s_1}}(\mathbf{x})$– satisfy the following:

- $\mathbf{s_2} \leftarrow \mathsf{Shr}^{\ell,p}_{\mathbf{s_1}}(h_\mathbf{k}(\mathbf{x}) \div p)$ and
- $y_\mathsf{r} = h_\mathbf{k}(\mathbf{x}) \mod p$

Moreover, by correctness of the secret sharing scheme, we see that $y_\mathsf{q} = h_\mathbf{k}(\mathbf{x}) \div p$. So, by the Quotient-Remainder Theorem, indeed $y = y_\mathsf{q} p + y_\mathsf{r}$ is equal to $h_\mathbf{k}(\mathbf{x})$. □

Proof of Theorem 3.2, tag size. Fix n, p. The tag $\mathsf{tag} = (\mathbf{s_2}, y_\mathsf{r})$ outputted by $\mathsf{Mac}^{n,p}$ resides in $\mathcal{S}_2 \times [0..p-1]$ for $\mathcal{S}_2$ as specified by the secret sharing scheme with which the MAC is instantiated.

When instantiated with the secret sharing scheme in Figure 3, $\mathcal{S}_2 = \{0,1\}^\ell \times [0..p-1]^\ell$, for $\ell = \left\lfloor \log_p(np^2) \right\rfloor$, and its size is

$$|\mathbf{s_2}| = \ell \cdot (|p| + 1)$$

where $|p| = \lfloor \log_2 p \rfloor + 1$ is the binary representation length of an element in $[0..p-1]$. So, the tag size is:

$$\begin{aligned}
|\mathsf{tag}| &= \ell \cdot (|p| + 1) + |p| \\
&= \left\lfloor \log_p(np^2) \right\rfloor \left(\lfloor \log_2 p \rfloor + 2 \right) + \lfloor \log_2 p \rfloor + 1 \\
&\approx \log_p(np^2) \left(\log_2 p + 2 \right) + \log_2 p + 1 \text{ (ignoring } \lfloor \cdot \rfloor) \\
&= \log_2(np^2) \left(1 + \frac{2}{\log_2 p} + \frac{\log_2 p + 1}{\log_2 n + 2\log_2 p} \right) \\
&\le \log_2(np^2) \left(1 + 1 + \frac{2}{2} \right) \text{ (assigning } p = 2, n = 1) \\
&= 3 \cdot \log_2(np^2)
\end{aligned}$$

The message is in $[0..p-1]^n$ so its size is

$$|\mathbf{x}| = n \cdot |p|$$

so the size ratio of tag vs. message goes to zero as n goes to infinity:

$$\begin{aligned}
\frac{|\mathsf{tag}|}{|\mathbf{x}|} &= \frac{\ell \cdot (|p| + 1) + |p|}{n \cdot |p|} \\
&\le \frac{3 \log_2(np^2)}{n \cdot \log_2 p} \\
&\overset{n\to\infty}{\to} 0
\end{aligned}$$

□

Proof of Theorem 3.2, unforgeability. Fix n, p. We prove that our MAC (Figure 4) is a $\frac{1}{p}$-one-time unforgeable MAC. The proof is via a reduction to the $\frac{1}{p}$-one-time unforgeability of the MAC in Figure 2. The reduction proceeds in three steps, gradually modifying the latter scheme (denoted $\mathcal{Q}_0$ in the following), going through two intermediate MAC schemes (denoted $\mathcal{Q}_1, \mathcal{Q}_2$), until reaching our MAC (denoted $\mathcal{Q}^3$), and while proving that $\Pr[\mathsf{Mac\text{-}Forge}^{\text{one-time}}_{\mathcal{A},\mathcal{Q}_i}(\lambda) = 1]$ is non-increasing when reducing from one scheme $\mathcal{Q}_i$ to the next $\mathcal{Q}_{i+1}$. Details follow.

$\mathcal{Q}_0$ is the MAC specified in Figure 2. That is:

- $\mathcal{Q}_0.\mathsf{Gen}$ outputs $\mathbf{k} \leftarrow_R \mathbb{Z}_p^{n+1}$.
- $\mathcal{Q}_0.\mathsf{Mac}_\mathbf{k}(\mathbf{x})$, given $\mathbf{x} \in \mathbb{Z}_p^n$, outputs $y_\mathsf{r} := h_\mathbf{k}(\mathbf{x})$ for $h_\mathbf{k} : \mathbb{Z}_p^n \to \mathbb{Z}$ defined, as in Equation 1, by $h_\mathbf{k}(\mathbf{x}) = k_0 + \sum_{i=1}^n k_i x_i \mod p$.
- $\mathcal{Q}_0.\mathsf{Vrf}_\mathbf{k}(\mathbf{x}, y_\mathsf{r})$, given $(\mathbf{x}, y_\mathsf{r}) \in \mathbb{Z}_p^n \times \mathbb{Z}_p$,outputs acc if-and-only-if $y_\mathsf{r} = h_\mathbf{k}(\mathbf{x})$ (rej otherwise).

By Theorem 2.1, $\mathcal{Q}_0$ is (information theoretically) $\frac{1}{p}$-one-time unforgeable. That is, for every adversary $\mathcal{A}$,

$$\Pr[\mathsf{Mac\text{-}Forge}^{\text{one-time}}_{\mathcal{A},\mathcal{Q}_0}(\lambda) = 1] \le \frac{1}{p} \tag{2}$$

$\mathcal{Q}_1$ is similar to $\mathcal{Q}_0$ except that the tag is augmented with a uniformly random element in $\mathcal{S}_2$ (for $\mathcal{S}_2$ being the space of the 2nd share in the secret sharing scheme). That is:

- $\mathcal{Q}_1.\mathsf{Gen}$ outputs $\mathbf{k} \leftarrow \mathcal{Q}_0.\mathsf{Gen}()$.
- $\mathcal{Q}_1.\mathsf{Mac}_\mathbf{k}(\mathbf{x})$, given $\mathbf{x} \in \mathbb{Z}_p^n$, samples uniformly random $\mathbf{s_2} \leftarrow_R \mathcal{S}_2$, computes $y_\mathsf{r} \leftarrow \mathcal{Q}_0.\mathsf{Mac}_\mathbf{k}(\mathbf{x})$, and outputs $(\mathbf{s_2}, y_\mathsf{r})$.
- $\mathcal{Q}_1.\mathsf{Vrf}_\mathbf{k}(\mathbf{x}, (\mathbf{s_2}, y_\mathsf{r}))$, given $(\mathbf{x}, y_\mathsf{r}) \in \mathbb{Z}_p^n \times \mathbb{Z}_p$, ignores $\mathbf{s_2}$ and outputs the output of $\mathcal{Q}_0.\mathsf{Vrf}_\mathbf{k}(y_\mathsf{r})$

We show that the probability of forging remains the unchanged. For this purpose we show that, given any adversary $\mathcal{A}_1$ for Q_1, we can construct an adversary $\mathcal{A}_0$ for Q_0 with the same forging probability. The adversary $\mathcal{A}_1$ is defined as follows:

(1) Upon receiving a message $\mathbf{x}' \in \mathbb{Z}_p^n$ from $\mathcal{A}_1$, $\mathcal{A}_0$ queries the oracle Q_0.Mac on $\mathbf{x}'$ to receive a tag y'_{r}, samples uniformly random $\mathbf{s_2}' \leftarrow_R \mathcal{S}_2$ and provides $\mathcal{A}_1$ with the tuple $(\mathbf{s_2}', y'_{\mathsf{r}})$.
(2) Upon receiving a tuple $(\mathbf{x}, (\mathbf{s_2}, y_{\mathsf{r}}))$ from $\mathcal{A}_1$, $\mathcal{A}_0$ outputs the received tuple.

Observe that the view of $\mathcal{A}_1$ when executed by $\mathcal{A}_0$ is identical to its view in $\text{Mac-Forge}^{\text{one-time}}_{\mathcal{A},Q_1,\mathcal{F}}(\lambda)$ (because the tags are identically distributed). Moreover, any (msg, tag) pair that passes Q_1.Vrf also passes Q_0.Vrf (because verification ignores the additional values in the tag). Therefore,

$$\Pr[\text{Mac-Forge}^{\text{one-time}}_{\mathcal{A},Q_1}(\lambda) = 1] = Pr[\text{Mac-Forge}^{\text{one-time}}_{\mathcal{A},Q_0}(\lambda) = 1] \tag{3}$$

Q_2 is similar to Q_1 except that Mac does not output random $\mathbf{s_2}$ but rather sets them to be the 2nd share in a secret sharing of $y_{\mathsf{q}} = (k_0 + \sum_{i=1}^n k_i x_i) \div p$ using a random 1st share $\mathbf{s_1} \leftarrow_R \mathcal{S}_1$. That is:

- Q_2.Gen outputs $\mathbf{k} \leftarrow Q_1.\text{Gen}()$.
- $Q_2.\text{Mac}_{\mathbf{k}}(\mathbf{x})$ samples uniformly random $\mathbf{s_1} \leftarrow_R \mathcal{S}_1$; computes:

$$y = k_0 + \sum_{i=1}^n k_i x_i \text{ (over the reals)}$$
$$y_{\mathsf{q}} := y \div p$$
$$y_{\mathsf{r}} := y \bmod p$$
$$\mathbf{s_2} \leftarrow \text{Shr}_{\mathbf{s_1}}(y_{\mathsf{q}}), \text{ where } \mathbf{s_1} \leftarrow_R \mathcal{S}_1;$$

  and outputs $(\mathbf{s_2}, y_{\mathsf{r}})$.
- Q_2.Vrf is identical to Q_1.Vrf.

The $\text{Mac-Forge}^{\text{one-time}}_{\mathcal{A},Q_i}$ experiments $i = 1, 2$ are identical except for the way $\mathbf{s_2}$ is generated in Q_i.Mac: in Q_1 it is sampled uniformly at random in $\mathcal{S}_2$, whereas in Q_2 it is the outputs of $\text{Shr}_{\mathbf{s_1}}$ on a uniformly random $\mathbf{s_1} \in \mathcal{S}_1$. By the premise that the secret sharing scheme has a uniform 2nd share, these distributions are identically, and so the output of the forging experiments is identically distributed, implying:

$$\Pr[\text{Mac-Forge}^{\text{one-time}}_{\mathcal{A},Q_2}(\lambda) = 1] = \Pr[\text{Mac-Forge}^{\text{one-time}}_{\mathcal{A},Q_1}(\lambda) = 1].$$

Q_3 is the MAC specified in Figure 4. Q_3 is similar to Q_2 except that: (1) sampling of $\mathbf{s_1} \in \mathcal{S}_1$ is done in Gen (rather than in Mac), and (2) Vrf verifies also that $\mathbf{s_2}$ is consistent with the key and message (rather than only verifying that y_{r} is consistent). That is:

- Q_3.Gen samples $\mathbf{k} \leftarrow Q_2.\text{Gen}()$ and $\mathbf{s_1} \leftarrow_R \mathcal{S}_1$, and outputs $(\mathbf{k}, \mathbf{s_1})$.
- $Q_3.\text{Mac}_{\mathbf{k},\mathbf{s_1}}(\mathbf{x})$ executes $Q_2.\text{Mac}_{\mathbf{k}}(x)$ while hardwiring the random values $\mathbf{s_1}$ to be the one provided as the 2nd part of the key.
- $Q_3.\text{Vrf}_{\mathbf{k},\mathbf{s_1}}(\mathbf{x}, (\mathbf{s_2}, y_{\mathsf{r}}))$ outputs acc if-and-only-if both the following holds for $y := k_0 + \sum_{i=1}^n k_i x_i$:

$$\text{Rec}_{\mathbf{s_1}}(\mathbf{s_2}) = (y \div p)$$
$$y_{\mathsf{r}} = y \bmod p$$

Observe that the only difference in logic between Q_3 and Q_2 is that, in Q_3, (msg, tag) pair given to Q_3.Vrf must satisfy an *additional constraint* on top of the constraint required in Q_2 (specifically, they must satisfy also they verification of $\mathbf{s_2}$ components in the tag). Therefore, the probability of forging can only decrease, namely:

$$\Pr[\text{Mac-Forge}^{\text{one-time}}_{\mathcal{A},Q_3}(\lambda) = 1] \le \Pr[\text{Mac-Forge}^{\text{one-time}}_{\mathcal{A},Q_2}(\lambda) = 1]$$

We conclude that our scheme of Figure 4 (denoted Q_3) satisfies that:

$$\begin{aligned}&\Pr[\text{Mac-Forge}^{\text{one-time}}_{\mathcal{A},Q_3}(\lambda) = 1]\\ &\le \Pr[\text{Mac-Forge}^{\text{one-time}}_{\mathcal{A},Q_0}(\lambda) = 1]\\ &\le \frac{1}{p}\end{aligned}$$

That is, the scheme is $\frac{1}{p}$-one time unforgeable. □

C.2 Supplemental Material for Section 3.2

We prove Proposition 3.2.

Proof. Let $\mathcal{E} = (\text{Gen}, \text{Enc}, \text{Dec}, \text{Eval})$ and $Q = (Q.\text{Gen}, \text{Mac}, \text{Vrf})$. We define $\widehat{\mathcal{E}} = (\widehat{\text{Gen}}, \widehat{\text{Enc}}, \text{Dec}, \widehat{\text{Eval}})$ as follows:

- $\widehat{\text{Gen}}(1^\lambda)$ samples $(pk, sk) \leftarrow \text{Gen}(1^\lambda)$ and a uniform message m^* in the message space, computes $c^* \leftarrow \text{Enc}_{pk}(m^*)$, and outputs $(\widehat{pk}, \widehat{sk}) = ((pk, c^*), sk)$.
- $\widehat{\text{Enc}}_{\widehat{pk}}(m)$, where $\widehat{pk} = (pk, c^*)$, outputs $\text{Enc}_{pk}(m)$.
- $\widehat{\text{Eval}}_{\widehat{pk}}(f; c_1, \ldots, c_n)$, where $\widehat{pk} = (pk, c^*)$, if $c_1 \neq c^*$, outputs $\text{Eval}_{pk}(f; c_1, \ldots, c_n)$; otherwise outputs $\text{Enc}_{pk}(\text{acc})$.

It is easy to show that $\widehat{\mathcal{E}}$ is semantically secure via a reduction to the semantic security of $\mathcal{E}$.

We define $\mathcal{A}$ as follows: given $\widehat{pk} = (pk, c^*)$, it submits the pair $(c_{\text{msg}}, c_{\text{tag}})$ defined by:

$$c_{\text{msg}} := c^* \text{ and } c_{\text{tag}} = \text{Enc}_{pk}(0).$$

Clearly $\mathcal{A}$ is ppt. We next show that $\mathcal{A}$ succeeds in the homomorphic forging experiment, i.e., $\text{Mac-Forge}_{\mathcal{A},Q,\widehat{\mathcal{E}}} = 1$. By definition of $\widehat{\text{Eval}}$, $\widehat{\text{Eval}}_{\widehat{pk}}(\text{Vrf}; c_{\text{msg}}, c_{\text{tag}})$ outputs $\text{Enc}_{pk}(\text{acc})$ (because $c_{\text{msg}} = c^*$). So,

$$\text{Dec}_{sk}[\widehat{\text{Eval}}_{\widehat{pk}}(\text{Vrf}; c_{\text{msg}}, c_{\text{tag}})] = \text{acc}.$$

Namely, $\text{Mac-Forge}_{\mathcal{A},Q,\widehat{\mathcal{E}}} = 1$ (where the $1 - \text{neg}$ probability in the theorem addresses schemes $\mathcal{E}$ with a negligible decryption failure probability, where the probability is over the randomness in Gen and Enc). □

Proof of Proposition 3.3

Proof. By definition, full composability guarantees correctness of homomorphic evaluation over arbitrary ciphertexts (rather than only the output of an honest execution of the encryption algorithm); cf. Definition 2.2. So, the full homomorphism of $\mathcal{E}$ w.r.t $\{\text{Vrf}\}$ guarantees that *for all* ciphertexts $c_{\text{msg}}, c_{\text{tag}}$ in the ciphertext space, with probability $1 - \text{neg}(\lambda)$ over the choice of $\mathsf{k} \leftarrow \text{MAC.Gen}(1^\lambda)$ and $(pk, sk) \leftarrow \text{HE.Gen}(1^\lambda)$, the following holds:

$$\text{Dec}_{sk}\left(\text{Eval}_{pk}\left(\text{Vrf}_{\mathsf{k}}; c_{\text{msg}}, c_{\text{tag}}\right)\right) = \text{Vrf}_{\mathsf{k}}\left(\text{Dec}_{sk}(c_{\text{msg}}), \text{Dec}_{sk}(c_{\text{tag}})\right). \tag{4}$$

This contradicts the (cleartext) unforgeability of $\mathcal{Q}$ (details follow).

Given any adversary $\mathcal{A}$ for the HE-unforgeability experiment, we construct an adversary $\mathcal{A}'$ for the (cleartext) unforgeability experiment, as follows. The adversary $\mathcal{A}'$, given $\mathsf{k} \leftarrow \mathsf{MAC.Gen}(1^\lambda)$, generates $(pk, sk) \leftarrow \mathsf{HE.Gen}(1^\lambda)$, and provides pk to $\mathcal{A}$. Upon receiving msg' from $\mathcal{A}$, $\mathcal{A}'$ forwards it to the challenger to receive a valid tag', and forwards the latter to $\mathcal{A}$. Upon receiving $(c_{\mathsf{msg}}, c_{\mathsf{tag}})$ from $\mathcal{A}$, $\mathcal{A}'$ decrypts using sk to obtain $\mathsf{msg} \leftarrow \mathsf{Dec}_{sk}(c_{\mathsf{msg}})$ and $\mathsf{tag} \leftarrow \mathsf{Dec}_{sk}(c_{\mathsf{tag}})$, and forwards $(\mathsf{msg}, \mathsf{tag})$ to the challenger. Then the following holds.

$$\begin{aligned}
&\Pr[\mathsf{Mac\text{-}Forge}^{\text{one-time}}_{\mathcal{A},\mathcal{Q}}(\lambda) = 1] \\
&= \Pr[\mathsf{Vrf}_{\mathsf{k}}\left(\mathsf{Dec}_{sk}(c_{\mathsf{msg}}), \mathsf{Dec}_{sk}(c_{\mathsf{tag}})\right) = \mathsf{acc}] \\
&= \Pr[\mathsf{Dec}_{sk}\left(\mathsf{Eval}_{pk}\left(\mathsf{Vrf}_{\mathsf{k}}; c_{\mathsf{msg}}, c_{\mathsf{tag}}\right)\right) = \mathsf{acc}] \\
&= \Pr[\mathsf{Mac\text{-}Forge}^{\text{one-time}}_{\mathcal{A},\mathcal{Q},\mathcal{E}}(\lambda) = 1]
\end{aligned}$$

(where the first equality follows from the definition of the forging experiment; the second – from full composability (cf. Equation 4); the third from the the definition of the HE-forging experiment; and the last step follows from the definition of $\mathcal{A}$). By the unforgeability of $\mathcal{Q}$, we conclude that $\rho = \mathsf{neg}(\lambda)$. Therefore, $\Pr[\mathsf{Mac\text{-}Forge}^{\text{one-time}}_{\mathcal{A},\mathcal{Q},\mathcal{E}}(\lambda) = 1] = \mathsf{neg}(\lambda)$ for every adversary $\mathcal{A}$. Namely, $\mathcal{Q}$ is HE-unforgeable. □

HE-correctness says that homomorphic evaluation of Vrf on freshly encrypted ciphertexts, yields the same result as evaluation in cleartext.

Definition C.1 (HE-correctness). *Let $\mathcal{E} = (HE.\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be a HE scheme, and $\mathcal{Q} = (\mathsf{MAC.Gen}, \mathsf{Mac}, \mathsf{Vrf})$ an $\mathcal{E}$-friendly MAC for domain A. $\mathcal{Q}$ is $\mathcal{E}$-correct, if for every $\mathsf{msg} \in \mathsf{A}$ and $\lambda \in \mathbb{N}$,*

$$\mathsf{Dec}_{sk}\left(\mathsf{Eval}_{pk}\left(\mathsf{Vrf}_{\mathsf{k}}; \mathsf{Enc}_{pk}(\mathsf{msg}), \mathsf{Enc}_{pk}(\mathsf{Mac}_{\mathsf{k}}(\mathsf{msg}))\right)\right) = \mathsf{acc}.$$

with probability $1 - \mathsf{neg}(\lambda)$ over the choice of $\mathsf{k} \leftarrow \mathsf{MAC.Gen}(1^\lambda)$ and $(pk, sk) \leftarrow HE.\mathsf{Gen}(1^\lambda)$.

Proposition C.1 (HE-correctness). *For every HE scheme $\mathcal{E}$ and HE- friendly MAC scheme $\mathcal{Q}$, if $\mathcal{Q}$ is correct then it is HE-correct.*

Proof. The proof follows immediately from the homomorphism of $\mathcal{E}$ (cf. Definition B.3, C-homomorphism). □

D Supplemental Material: Section 4

D.1 Protocols: Further Details

Next, we explain how to extend our protocols for addressing HE schemes that are not fully composable; how to instantiate our protocol to support either real value data or elements in a finite field $\mathbb{Z}_p^n$ (rather than integers); finally we discuss extensions for addressing malicious data consumers.

Remark D.1 (full composability via bootstrapping). *Our proof of soundness against malicious* dataKeeper *requires that the HE to be fully composable w.r.t $\{Rec, \mathsf{Vrf}\}$. We can guarantee the latter via bootstrapping (cf. [41]) by modifying Step 3 in* retrieve *to output (c, c_σ) computed by:*

$$\begin{aligned}
c &\leftarrow \mathsf{Eval}_{pk}(Rec_{s_1}; bootstrap_{pk}(c_2)) \\
c_\sigma &\leftarrow \mathsf{Eval}_{pk}(\mathsf{Vrf}_{\mathsf{k}}; c_2, bootstrap_{pk}(c_{\mathsf{tag}})).
\end{aligned}$$

Our protocol is generic and can be instantiated with any secret sharing, MAC and HE schemes that satisfy the properties listed in Figure 6, Common Parameters. Examples follow.

Corollary D.2 (our protocol for reals). *The protocol in Figure 6 can be instantiated with our secret sharing scheme and MAC schemes (cf. Figures 3-4) together with any HE that supports additive homomorphism over the reals and is fully composable or bootstrappable w.r.t $\{Rec, \mathsf{Vrf}\}$. E.g., CKKS [21] is bootstrappable (and might be fully composable, to the best of our knowledge). In this instantiation, the storage size for storing $x \in [0..p-1]^n$ is:*

$$|\mathsf{storage}(x; index, \lambda)| = |x| + O\left(\log_2(np^2)\right)$$

i.e., the asymptotic authentication overhead approaches zero:

$$\lim_{n\to\infty} \frac{|\mathsf{storage}(x; index, \lambda)| - |x|}{|x|} = \lim_{n\to\infty} \frac{O\left(\log_2(np^2)\right)}{O(n \log_2 p)} = 0$$

Corollary D.3 (our protocol for $\mathbb{Z}_p^n$). *The protocol in Figure 6 can be instantiated with additive secret sharing. In additive secret sharing for $\mathbb{Z}_p^n$; $\mathsf{Shr}(x)$ outputs uniformly random $\mathbf{s_1} \in \mathbb{Z}_p^n$ and $\mathbf{s_2} = \mathbf{x} - \mathbf{s_1} \bmod p$; and $Rec(\mathbf{s_1}, \mathbf{s_2})$ outputs $\mathbf{s_1} + \mathbf{s_2} \bmod p$. Clearly, this scheme has a random 1st share, uniform 2nd share, and is friendly with respect to any additive-homomorphic encryption over $\mathbb{Z}_n^n$. and MAC for $\mathbb{Z}_p^n$ (cf. Figure 2) together with any HE that supports additive homomorphism over $\mathbb{Z}_p^n$ and is fully composable or bootstrappable w.r.t $\{Rec, \mathsf{Vrf}\}$. E.g., BGV [18], B/FV [17, 29] and TFHE [22] are bootstrappable (and might be fully composable, to the best of our knowledge). In this instantiation, the storage size for storing $x \in \mathbb{Z}_p^n$ is:*

$$|\mathsf{storage}(x; index, \lambda)| = |x| + (\lfloor \log_2 p \rfloor + 1)$$

i.e., the asymptotic authentication overhead approaches zero:

$$\lim_{n\to\infty} \frac{|\mathsf{storage}(x; index, \lambda)| - |x|}{|x|} = \lim_{n\to\infty} \frac{O\left(\log_2 p\right)}{O(n \log_2 p)} = 0$$

For simplicity of exposition, we present our protocol (Figure 6) for the case of storing of a single $(\mathsf{msg}, \mathsf{tag})$ (albeit, msg can be a vector of data items). Our protocol extends to retrieve many messages and tags, while compressing the verification outputs to a single authenticity indicator ciphertext decrypting to acc if-and-only-if all pairs are verified. This extension is explained below.

Remark D.4 (Correctness with Malicious Data Consumer.). *Throughout this work our focus is addressing* malicious data keeper *to guarantee data integrity. In this remark we discuss protection against* malicious data consumers*, to ensure that they execute only authorized computations on authorized data, can be provided using generic proof techniques for verifiable homomorphic computation (vFHE). Concretely, data consumers will generate a proof that they performed the intended homomorphic computation, including MAC verification, on the inputs provided by the data keeper. The proof together with the ciphertexts containing the computation result and the authenticity indicator, are submitted to the decrypting entity and verified prior to decryption.*

D.2 Protocols: Complexity and Optimizations

The implementation of our system performs storage and communication optimizations as described below.

Denote by $u = \lceil \frac{n}{\text{slots}} \rceil$.

In the unbatched mode for 72-bits information theoretic security we generate the tag 4 times, using independent randomness with the following overall complexity:

- *Storage size:* dataProd stores $s_2 = s_{\mathsf{q}}^{\oplus} \cdot p + s_{\mathsf{r}}$ and $\mathsf{tag} = s_{\mathsf{q}^{\oplus}}^{\mathsf{tag}} \cdot p^2 + s_{\mathsf{r}}^{\mathsf{tag}} \cdot p + y_0$ for each tag, where each MAC is computed on $(s_{\mathsf{q}}^{\oplus}, s_{\mathsf{r}})$. Overall, we store a single double for each data item and another double for each tag of each data item.
- *Communication size:* dataKeeper downloads the stored s_2 and tags and computes: $(s_{\mathsf{q}}^{\oplus}, s_{\mathsf{r}})$, and $\mathsf{tag}_{\mathsf{r}} = s_{\mathsf{r}}^{\mathsf{tag}} \cdot p + y_0$ and $s_{\mathsf{q}^{\oplus}}^{\mathsf{tag}}$ for each tag, and HE-encrypts the computed values. Overall, 10 ciphertexts are transmitted to dataConsmr. For retrieval of n (unbatched) data items we utilize the ciphertexts' slots, resulting in $10{\cdot}u$ ciphertexts transmitted to dataConsmr.
- *HE-operations:* dataConsmr performs overall $14{\cdot}u$ multiplications: u in Rec, $3 \cdot u$ in each Vrf, and another u for squaring the output of each Vrf to obtain the compressed correctness indicator. Since Rec and Vrf are degree-1 polynomials overall it requires HE parameters to support a multiplicative depth 2 due to the squaring done for the correctness indicator aggregation (square diff).

In the batched mode, we compute batched tags for n data items. To achieve at 72 bits information theoretic security we generate a tag 6 times, using independent randomness, obtaining the following overall complexity:

(1) *Storage size:* dataProd computes MAC on batches of size u on $\{(s_{\mathsf{q}^{\oplus},i}, s_{\mathsf{r},i})\}_{i\in[n]}$ and stores:
   - $\forall i \in [n]$: $s_{2,i} = s_{\mathsf{q}^{\oplus},i} \cdot p + s_{\mathsf{r},i}$
   - $\forall j \in [\text{slots}]$: $\mathsf{tag}_{\mathsf{r},j} = s_{\mathsf{r},2,j}^{\mathsf{tag}} \cdot p^2 + s_{\mathsf{r},1,j}^{\mathsf{tag}} \cdot p + y_0$ and
   $$\mathsf{tag}_{\mathsf{q}^{\oplus},j} = (s_{\mathsf{q}^{\oplus},2,j}^{\mathsf{tag}}, s_{\mathsf{q}^{\oplus},1,j}^{\mathsf{tag}})$$
   That is, in each tagging repetition we produce slots tags and store a single double plus two bits incorporated into a single stored per tag, overall done 6 times. In addition, we store a single double for each data item.

(2) *Communication size:* dataKeeper downloads the stored $\{s_{2,i}\}_{i\in[n]}$ and the corresponding tags it computes $\{(s_{\mathsf{q}^{\oplus},i}, s_{\mathsf{r},i})\}_{i\in[n]}$ and HE-encrypts each element (of each share and of each tag) with full utilization of the ciphertext slots. Overall, 3 ciphertexts are required for the components batched tag generated in each repetition, and thus, for n batched data items $2 \cdot u{+}18$ ciphertexts are transmitted to dataConsmr.

(3) *HE-operations:* dataConsmr performs overall $13{\cdot}u{+}12$ HE multiplications : u in Rec, and $2 \cdot u + 2$ in each Vrf. Since Rec and Vrf are degree-1 polynomials the overall computation only requires HE parameters to support multiplicative depth 1.

D.3 Proof of Theorems 4.1 and 4.2

First, we prove Theorem 4.1 .

Proof. Let λ be a security parameter and $\mathcal{E} = (\mathsf{HE.Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be a C-homomorphic CPA-secure HE scheme that is fully composable w.r.t the verification algorithm Vrf of the MAC scheme $\mathcal{Q}$ specified next. A PRF family $\{f_z : \{0,1\}^* \to \mathcal{K}\}_{z\in\{0,1\}^\lambda}$ and a random key MAC $\mathcal{Q} = (\mathsf{MAC.Gen}, \mathsf{Mac}, \mathsf{Vrf})$ that is $\mathcal{E}$-friendly.

The polynomial-time complexity of dataConsmr, dataKeeper and the party executing the trusted setup is straightforward as all the components and computations are polynomial time.

The *correctness* property stems from $\mathcal{Q}$ being HE-correct according to Proposition C.1, pseudorandomness of $\{f_z : \{0,1\}^* \to \mathcal{K}\}$, and from the correctness of $\mathcal{E}$. More formally, by definition

$$\begin{aligned}&\Pr[\mathsf{Dec}_{sk}(c_x, c_\sigma) \neq (x, \text{acc})]\\ &= \Pr[\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(x), \mathsf{Eval}_{pk}(\mathsf{Vrf}_{\mathsf{k}}; \mathsf{Enc}_{pk}(x), \mathsf{Enc}_{pk}(\mathsf{tag})) \neq (x, \text{acc})]\end{aligned}$$

Where the probability is taken over the choice of $z \leftarrow_R \{0,1\}^\lambda$, $\mathsf{k} \leftarrow f_z(index)$ and $(pk, sk) \leftarrow \mathsf{HE.Gen}(1^\lambda)$

Then it holds,

$$\begin{aligned}&\Pr[\mathsf{Dec}_{sk}(c_x, c_\sigma) \neq (x, \text{acc})]\\ &\leq \mathsf{neg}(\lambda) +\\ &\Pr[\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(x), \mathsf{Eval}_{pk}(\mathsf{Vrf}_{\mathsf{k}}; \mathsf{Enc}_{pk}(x), \mathsf{Enc}_{pk}(\mathsf{tag})) \neq (x, \text{acc})]\\ &< \mathsf{neg}(\lambda)\end{aligned}$$

Where the probability is taken over the choice of $\mathsf{k} \leftarrow \mathsf{MAC.Gen}(1^\lambda)$ and $(pk, sk) \leftarrow \mathsf{HE.Gen}(1^\lambda)$.

The *privacy* for any ppt adversary dataConsmr* relies on its view being independent of x (and presumably only depending on its size). Concretely, due to CPA-security of $\mathcal{E}$ and k depending only on $f_z(\cdot)$ (i.e., independent of x), we have for any *index* and λ,

$$(\mathsf{Enc}_{pk}(x), \mathsf{Enc}_{pk}(\mathsf{tag}), index, \lambda) \approx_c (\mathsf{Enc}_{pk}(x^*), \mathsf{Enc}_{pk}(\mathsf{tag}^*), index, \lambda)$$

for any $x^* \in \mathsf{A}$ and $\mathsf{tag}^* \in \mathsf{Tgs}$ s.t $|x| = |x^*| \wedge |\mathsf{tag}| = |\mathsf{tag}^*|$. Moreover, since $\mathsf{view}_{\mathsf{dataConsmr}^*}(x)$ consists exactly of

$$(\mathsf{Enc}_{pk}(x), \mathsf{Enc}_{pk}(\mathsf{tag}), index, \lambda)$$

we obtain the desired for any ppt distinguisher $\mathcal{D}$.

The *soundness* property against any ppt dataKeeper* stems from the HE-unforgeability of $\mathcal{Q}$ and the pseudorandomness of $\{f_z : \{0,1\}^* \to \mathcal{K}\}$. More formally, given that $\mathcal{E}$ is C-homomorphism ,and fully composable w.r.t. verification algorithm $\{\mathsf{Vrf}\}$ and $\mathcal{Q}$ is $\mathcal{E}$-friendly it follows from Proposition 3.3 that $\mathcal{Q}$ is HE-unforgeable. First, we note that since $\mathcal{Q}$ is HE-unforgeable w.r.t random key $\mathsf{k} \leftarrow \mathsf{MAC.Gen}(1^\lambda)$ the it is also HE-unforgeable w.r.t pseudorandom key $\mathsf{k} \leftarrow f_z(index)$ for $z \leftarrow_R \{0,1\}^\lambda$.

Next we assume towards contradiction that soundness does not hold and construct an adversary that violates the HE-unforgeable of $\mathcal{Q}$. That is, assume there exists a ppt dataKeeper* and $(x, index)$ tuple such that,

$$\Pr[\mathsf{Dec}_{sk}(c_x, c_\sigma) = (x', \text{acc})] > \frac{1}{p(\lambda)}$$

for some polynomial $p(\cdot)$ and any $x' \neq x$. We construct an adversary $\mathcal{A}$ for the $\mathsf{Mac\text{-}Forge}^{\text{one-time}}_{\mathcal{A},\mathcal{Q},\mathcal{E}}$ experiment that runs internally dataKeeper* and proceeds as follows:

(1) upon receiving pk it submits to the challenger x and receives tag on x.
(2) it forwards to dataKeeper* the tuple (x, tag)
(3) once dataKeeper* outputs (c_x, c_{tag}) it forwards then to the challenger.

Since the view of dataKeeper* is identical when executed internally by $\mathcal{A}$ to its run in oneSidedPriv we obtain the following,

$$\begin{aligned}
&\Pr[\text{Mac-Forge}^{\text{one-time}}_{\mathcal{A},Q,\mathcal{E}}(\lambda) = 1] \\
&= \Pr_{\mathcal{A}\text{ coins}}[\text{Dec}_{sk}(c_x, c_\sigma) = (x', \text{acc})] \\
&= \Pr_{\text{dataKeeper}^*\text{ coins}}[\text{Dec}_{sk}(c_x, c_\sigma) = (x', \text{acc})] \\
&> \frac{1}{p(\lambda)}
\end{aligned}$$

as desired.

□

For the proof of Theorem 4.2 we specify the required extensions from the proof of Theorem 4.1 for the common properties of correctness, polynomial time complexity, soundness and privacy against any ppt dataConsmr*. In the protocol twoSidedPriv we are utilizing in addition an $\mathcal{E}$-friendly 2-out-of-2 secret sharing scheme $\mathcal{S} = (\text{Shr}_{s_1}, \text{Rec}_{s_1})$ for a domain A with a random 1st share $s_1 \in \mathcal{S}_1$ and a uniform 2nd share $s_2 \in \mathcal{S}_2$ (cf. Definition 2.3).

Proof sketch. All entities are polynomial time correctness holds by the same reasoning as in oneSidedPriv, combined with the secret sharing scheme $\mathcal{S}$ being polynomial time and correct.

The soundness against dataKeeper* holds due to the full composability of $\mathcal{E}$ on $\text{Eval}_{pk}(\text{Rec}_{s_1}; c_2)$. That is, if $\text{Dec}_{sk}(\text{Eval}_{pk}(\text{Rec}_{s_1}; c_2)) \neq x$ then it implies $\text{Dec}_{sk}(c_2) \neq s2$ and hence the same argument as in oneSidedPrivcan be applied with respect to the success probability of any dataKeeper* providing an encrypted tag that homomorphically verifies on c_2 w.r.t the MAC key k.

The privacy against dataConsmr* holds similarly to oneSidedPriv with only difference of dataConsmr* receiving an encrypted secret share c_2 and a cleartext share s_1. From the CPA-security of $\mathcal{E}$ and s_1 being independent of x, the same argument as in oneSidedPriv applies here.

First note that the pseudorandomness property of f_z guarantees that for every x, the shares (s1,s2) produced by $f_z(\cdot)$ are distributed computationally close to shares produced MAC.Gen$(\cdot)$. Therefore, privacy against dataKeeper* stems from the latter together with the perfect security property of $\mathcal{S}$ and the fact that tag calculated only on s_2 and is independent of s_1.