HEAD: an FHE-based Outsourced Computation Protocol with Compact Storage and Efficient Computation

Lijing Zhou
Huawei Technology
zhoulijing@huawei.com

Ziyu Wang
Huawei Technology
wangziyu13@huawei.com

Xiao Zhang
Huawei Technology
zhangxiao81@huawei.com

Yu Yu
Shanghai Jiao Tong University
yuyu@cs.sjtu.edu.cn

Abstract

Fully homomorphic encryption (FHE) provides a natural solution for privacy-preserving outsourced computation, but a straightforward FHE-based protocol may suffer from high computational overhead and large ciphertext expansion rate, especially for computation-intensive tasks over large data, which are the main obstacles towards practical outsourced computation. In this paper, we present HEAD, a generic outsourced computation protocol that can be based on most mainstream (typically a BGV or GSW style scheme) FHE schemes with more compact storage and less computational costs than the straightforward FHE-based counterpart. In particular, our protocol enjoys a ciphertext/plaintext expansion rate of 1 (i.e., no expansion) at the server side. This is achieved by means of "pseudorandomly masked" ciphertexts, and the efficient transformations of them into FHE ciphertexts to facilitate privacy-preserving computation. Depending on the underlying FHE in use, our HEAD protocol can be instantiated with the three masking techniques, namely modulo-subtraction-masking, modulo-division-masking, and XOR-masking, to support the decimal integer, real, or binary messages. Thanks to these masking techniques, various homomorphic computation tasks are made more efficient and less noise accumulative. We evaluate the performance of our protocol on BFV, BGV, CKKS, and FHEW schemes based on the PALISADE and SEAL libraries, which confirms the theoretical analysis of the reduction computation costs and noise. For example, the computation time in our BFV tests in an x86 server for the sum or product of eight ciphertexts is reduced from 336.3 ms to 6.3 ms, or from 1219.4 ms to 9.5 ms, respectively. Furthermore, our multi-input masking and unmasking operations are more flexible than the FHE SIMD-batching, by supporting an on-demand configuration of FHE during each outsourced computation request.

1 Introduction

As an advanced form of encryption, homomorphic encryption (HE) allows anyone not in possession of the decryption key to perform certain computations (typically by evaluating a Boolean/arithmetic circuit) on ciphertexts, and produces the desired computation result in an encrypted form. It finds many useful privacy-preserving applications in the big data era. For example, a thin client can outsource his encrypted data and the following computation based on the data, to a more powerful cloud server, without revealing any substantial information.

An HE is called additively (resp., multiplicatively or fully) homomorphic if addition (resp., multiplication or both) is supported over ciphertexts. For instance, Paillier [37] is an additive HE that supports only homomorphic additions and scalar multiplications (but not multiplications between ciphertexts). A leveled HE is a slightly weaker form of FHE that supports arbitrary computations up to a certain depth that must be specified in advance so as to the parameter configuration.

In a leveled HE scheme, a ciphertext would accumulate some noise during homomorphic computations, and may not be correctly decrypted when the computations are beyond the preset level, since the introduced noise is too large and conceals the plaintext. In order to avoid a decrypt-then-re-encrypt naive noise reduction, a mechanism named bootstrapping is proposed, which generally evaluates a decryption circuit homomorphically [23, 25]. However, the bootstrapping overhead is considerable. Currently, a learning-with-errors (LWE) based leveled HE with a relatively efficient bootstrapping is the mainstream to achieve FHE when the system parameters are decoupled with the functions to be evaluated [32]. The rest of the paper would view a leveled HE scheme with bootstrapping as an FHE scheme.

Except for the FHE schemes based on the approximate greatest common division problem [13], the mainstream FHE construction relies on the LWE-hard problem. The FHE standardization procedure [5] and outstanding open-source libraries [1–4, 16, 31, 38] mainly focus on the LWE path [1]. Various LWE-based FHE schemes would like to support different types of input messages, rendering the BGV style and the GSW style. A BGV-style scheme, e.g., BGV [9] or

[1] Microsoft leads the FHE standardization based on the LWE path, which attracts a lot of attention from famous enterprises, institutes and universities.

BFV [10, 21], aims to handle the integer inputs (word-wise). CKKS [14] further extends an FHE to a real number input message. The GSW style copes with bit-wise binary messages, such as the GSW [29], FHEW [20], and TFHE [15] schemes. The homomorphic computation results in these schemes (except for CKKS) are precise, while CKKS may approximately evaluate a computation at a preset precision.

Although these mainstream schemes [9, 14, 20, 21] are implemented in open-source libraries [3, 16, 38], an FHE scheme is usually regarded as impractical in storage and computation. The storage problem lies in the notable FHE ciphertext expansion. An LWE-based FHE ciphertext, corresponding to an integer input, occupies several hundreds of kilobytes storage, since a large enough lattice dimension is a precondition of the security of the LWE-hard problem. Hence, the ciphertext expansion rate, i.e., the ratio of the FHE ciphertext size to the input message size, is quite large in an FHE scheme, which vastly limits an FHE application no matter for transferring or storing the ciphertext. Another focus for an FHE scheme is the inefficient computation concern. For a software implementation, a ciphertext-wise multiplication is usually the slowest operation in an integer FHE scheme, taking more than 100 ms. For a binary FHE scheme, the slowest computation unit is a ciphertext-wise XOR/XNOR, taking more than 1 second.

1.1 Related Work

Recently, some researches aim to cope with the significantly large ciphertext expansion rate and the low computation performance of an FHE scheme. These attempts, e.g., theoretically designing a compression algorithm or a software/hardware implementation technique, try to pursue compact storage and efficient computation for an FHE scheme.

Storage optimization. Packing [8] is a useful compression technique for an FHE ciphertext. When the encryption parameters satisfy some conditions, a bunch of input messages could be embedded to one ciphertext, rendering a relatively small ciphertext expansion rate that is amortized. The state-of-the-art packing works achieving a ciphertext expansion rate of around $10\times$ or $100\times$ for a BGV-style or a GSW-style FHE scheme [8, 35], respectively, which is not compact.

Gentry and Halevi [24] and Brakerski et al. [7] innovate some mechanisms to compress many GSW-style ciphertexts (occupying a large storage space) to a small-size ciphertext. The resulting ciphertext expansion rate for a compressed ciphertext is almost 1, with a price of losing some homomorphic functionalities in the compressed ciphertext.

Gentry et al. [27, 28] store an encrypted message in an AES-128 ciphertext, also rendering a ciphertext expansion rate of almost 1 if the input message size is divided by 128 bits [2]. This AES ciphertext could be converted to a BGV FHE ciphertext. This transformation enlightens the following researchers (including us) to overcome FHE disadvantages in a transformative way. Although employing several hardware acceleration tricks to enhance parallel processing, transforming a 120-block AES-128 ciphertext costs more than six minutes. Also, this transformed FHE ciphertext corresponding to the elements in the field of GF(2^d) has limited usages. The computation rules in this field are different from those in the integer field, which may not satisfy many applications.

The key-switching and modulus-switching are another two techniques related to decreasing the ciphertext size [28]. After multiplying two FHE ciphertexts, the dimension of the resulting ciphertext is increased, which requires a relinearization operation, i.e., a key-switching operation [26], to recover the ciphertext dimension. However, the size of a key-switched ciphertext does not decrease compared with the original size. Modulus-switching, proposed by Brakerski and Vaikuntanathan [10], is a noise management technique intentionally. Generally, when the noise of a ciphertext is reduced via modulus-switching, the ciphertext size gets smaller. However, the switched ciphertext still keeps a large expansion rate and the computation depth it could support is decreased.

Chen et al. [12] present a unidirectional transformation from an LWE-based FHE ciphertext to a Ring LWE-based one, which supports packing and has an amortized ciphertext size. Still, the ciphertext expansion rate keeps large.

The SEAL library integrates the `zlib` or `zstandard` compression libraries to further compress a serialized FHE component. Note that these storage optimizations are not in theory.

Computation optimization. Based on packing, the work from Smart and Vercauteren [41] enhances the evaluation strength for one packed ciphertext, i.e., the SIMD (single instruction, multiple data) batching. Besides storage optimization, the evaluation on a batched FHE ciphertext is equally acting on each element in that batch. A heavy computation burden is amortized to each message, which is an acceleration in some sense. A batched FHE scheme already offers both the storage and computation optimization in a practical application like a password leakage searching [36].

Aubry et al. [6] propose a circuit re-writer to highly decrease the depth of a Boolean/arithmetic circuit, which optimizes the computation, since the computational overhead of an FHE scheme highly relies on the circuit depth.

Another technical route about computation optimization of an FHE scheme is based on a hardware accelerator. There are a line of FPGA-based works [17–19, 40, 42] and some ASIC-based works [33, 39]. Especially, the hardware accelerator proposed by Feldmann et al. [22] outperforms state-of-the-art software implementations by up to 17,000×.

To the best of our knowledge, the related works achieving compact storage do not optimize the computation or even come at a price. Although the ciphertext expansion rate is almost 1, a compressible FHE ciphertext [7, 24] or

[2] Usually, the AES-128 encryption would pad the length of a plaintext to a multiple of 128 bits.

Table 1: HEAD contributions. (Ciphertext expansion rate abbreviated as CER)

HEAD protocols	**ModSub masking**	**ModDiv masking**	**XOR masking**	**XOR masking**
Input type	Decimal integer	Decimal integer	Real	Binary
Scheme	BFV/BGV	BFV/BGV	CKKS	FHEW
Storage optimization	CER = 1	CER = 1	CER = 1	CER = 1
Computation optimization	Many Adds or ScalarMult → one pt + ct	Many Mults → one ScalarMult	N.A.	Many XORs/XNORs → one NOT

an AES-transformed FHE ciphertext [28] sacrifices the homomorphic computation strength. Moreover, the packing or SIMD-batching solution trades off the FHE flexibility to amortized storage and computation optimization. When an application generates several ciphertexts in different time points, these ciphertexts may not be packed or batched. Some FHE schemes, such as TFHE [15], only support compress-used packing instead of SIMD-batching. In addition, the software and hardware optimization techniques are orthogonal to our work.

1.2 Our Contribution

We present HEAD [3], an FHE-based outsourced computation protocol, which aims to address/mitigate the ciphertext expansion and computational overheads (due to the use of FHE) at the server side by paying reasonable costs in communication. The protocol is generic and supports the transformation from many typical mask-with-pseudorandom encryptions to mainstream FHE (e.g., BGV-style or GSW-style) ciphertexts with IND-CPA security (as expected from FHEs). The protocol has the following advantages:

- **Generic.** The protocol can be instantiated with many FHE schemes. We apply the corresponding suitable masking methods leading to the modulo-subtraction-masking (ModSub), modulo-division-masking (ModDiv), and XOR-masking protocols, supporting different data types (integer, real or binary) of an underlying FHE.

- **Compact.** All FHE instantiations of the protocol achieve a ciphertext expansion rate of 1 for a long-term outsourced storage at the server side.

- **Efficient.** Our protocol optimizes various outsourced computing tasks (as summaried in Table 1). These optimizations reduce a large amount of time and/or noise. Reducing noise for an FHE scheme eliminates the need for a heavy bootstrapping, potentially reducing computation time.

  – The ModSub protocol optimizes one/more ciphertext-wise additions (Adds), or a scalar multiplication (ScalarMult) to only a single plaintext-ciphertext addition, i.e., many Adds or ScalarMult → one pt + ct.

  – The ModDiv protocol optimizes one/more ciphertext-wise multiplications (Mults) to only a single scalar multiplication, i.e., many Mults → one ScalarMult.

  – The XOR-masking protocol optimizes one/more ciphertext-wise XOR and XNOR operations (XORs/XNORs) to a single plaintext-ciphertext XOR, which is further optimized to a single ciphertext-wise NOT, i.e., many XORs/XNORs → one NOT.

- **Flexible.** Our flexibility lies in two aspects.

  – It is flexible to configure the FHE encryption parameters of a transformed FHE ciphertext according to (and just before) the computation. That is, the parameter choices do not need to be decided before the masking operation, i.e., the storage phase.

  – When a computation involves several stored masked messages, the retrieval does not have a limitation, and the selected masked messages can be transformed into a single batched FHE ciphertext (if batching is supported by the underlying FHE scheme) for subsequent evaluations.

Finally, we implement our protocols with the BFV, BGV, CKKS, and FHEW FHE schemes in PALISAD and SEAL libraries and compare them with a straightforward FHE-based protocol (without our techniques) as a baseline. Our evaluation shows that a masked ciphertext with the expansion rate of 1 occupies much less space compared with a straightforward FHE ciphertext, and the computation optimizations are significant. For example, in a typical encryption parameter setting, the expansion rate of a BFV ciphertext is around 200× or 20× when our protocol is not used in an unbatched or batched case, respectively. ModSub or ModDiv optimizes the sum or product of eight BFV ciphertexts from 336.3 ms to 6.3 ms, or from 1219.4 ms to 9.5 ms, respectively.

Paper organization. Section 2 introduces the notation, pseudorandom function (PRF) and FHE definitions, and a straightforward FHE-based outsourced computation scenario. Section 3 introduces the generic FHE-based outsourced computation protocol with compact storage and efficient computation,

[3] The name comes from HE combining with PAD.

and proves its security. Section 4 instantiates the generic HEAD protocol by three different masking techniques, i.e., modulo-subtraction, modulo-division, and XOR. We show that these instantiated protocols not only achieve a ciphertext expansion rate of 1, but also optimize various computations in different FHE schemes. Section 5 evaluates the performance of the three protocols based on the BFV, BGV, CKKS, and FHEW FHE schemes. Finally, Section 6 concludes this paper.

2 Preliminaries

2.1 Notation

$x \leftarrow_{\$} \mathcal{X}$ or $x \leftarrow \mathcal{X}$ refers to drawing an element x from the space $\mathcal{X}$ uniformly at random, or according to the distribution of $\mathcal{X}$, respectively. $\{x_i\}$ represents a set having I messages, and each message in this set x_i has an index i. The m-bit message x could be represented in binary as $x^0, \cdots, x^{m-1}$. $\widetilde{x}$ represents the masked message of x, using a pseudorandom r. $\oplus$ or $\odot$ is an XOR or XNOR operation, respectively.

In an FHE scheme, let sk, pk, $\mathsf{ct}(\cdot)$, $\mathcal{M}$, and $\mathcal{CT}$ denote a secret key, a public key, and a ciphertext, a message space, and a ciphertext space, respectively. Γ is a circuit to be evaluated in an FHE-based outsourced computation. λ is the security parameter. n, t, and Q are FHE encryption parameters for the ciphertext ring dimension, the plaintext modulus, and the ciphertext modulus, respectively.

2.2 Pseudorandom Function

Let PRF: $\mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ be an efficiently computable function. PRF is a pseudorandom function if for any probabilistic polynomial time (PPT) adversary $\mathcal{A}$ the advantage $\mathrm{Adv}^{\mathrm{PRF}}_{\mathcal{A}}(\lambda)$

$$\left| \Pr[b = 1 | \mathsf{key} \leftarrow_{\$} \mathcal{K}; b \leftarrow \mathcal{A}^{\mathrm{PRF}(key,\cdot)}] - \Pr[b = 1 | b \leftarrow \mathcal{A}^{\mathrm{RF}(\cdot)}] \right|$$

is negligible (negl), where $\mathsf{RF}(\cdot)$ is a random function RF : $\mathcal{X} \rightarrow \mathcal{Y}$. In the following paper, we use $r_i \leftarrow \mathsf{PRF}(\mathsf{key}, i)$ to instantiate the PRF function, which takes inputs on $\mathsf{key} \in \mathcal{K}$ and an index $i \in \mathcal{X}$, and outputs a pseudorandom number $r_i \in \mathcal{Y}$. We instantiate $\mathcal{Y}$ as same as the message space $\mathcal{M}$.

2.3 Homomorphic Encryption

We summarize the key components of an FHE scheme including four PPT algorithms, i.e., FHE = (KeyGen, Encrypt, Decrypt, Eval) in a high level.

- $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^{\lambda})$: This algorithm inputs a security parameter λ and outputs a secret key sk and a public key pk. pk implicitly refers to a public encryption key, a public relinear key [4], and a public Galois key for rotating an SIMD-batched ciphertext [34].

[4] After a ciphertext multiplication, the dimension of the ciphertext may increase. A relinearization is to recover the initial ciphertext dimension.

- $\mathsf{ct}(x) \leftarrow \mathsf{Encrypt}(\mathsf{pk}/\mathsf{sk}, x)$ and $x \leftarrow \mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}(x))$: The encryption and decryption algorithms convert a message $x \in \mathcal{M}$ to a ciphertext $\mathsf{ct}(x) \in \mathcal{CT}$ and vice versa. Note that an FHE scheme may support a symmetric or an asymmetric encryption mode, so that the encryption algorithm would require a secret key or a public key, respectively. We denote a set of ciphertexts or an SIMD-batched ciphertext by $\{\mathsf{ct}(x_i)\}$ or $\mathsf{ct}(\{x_i\})$, respectively.

- $\mathsf{ct}(\Gamma(x)) \leftarrow \mathsf{Eval}(\mathsf{pk}, \Gamma, \mathsf{ct}(x))$: A circuit Γ to be evaluated in an FHE scheme generally applies to the input $\mathsf{ct}(x)$ rendering $\mathsf{ct}(\Gamma(x))$. It is possible that the evaluation acts on all elements in the batched ciphertext, or parts of the inputs are in plaintext, both of which are denoted by $\mathsf{ct}(\Gamma(\{x_i\}))$.

Definition 1 (Correctness) *A fully homormophic Encryption scheme FHE is correct if for all $x \in \mathcal{M}$, and $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^{\lambda})$, the following probability holds:*

$$\Pr\left[\begin{array}{l} \mathsf{Decrypt}(\mathsf{sk}, \\ \mathsf{Encrypt}(\mathsf{pk}/\mathsf{sk}, x)) = M; \\ \mathsf{Decrypt}(\mathsf{sk}, \mathsf{Eval}(\mathsf{pk}, \Gamma, \\ \mathsf{Encrypt}(\mathsf{pk}/\mathsf{sk}, x))) = \Gamma(x) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Informally, an encryption scheme is indistinguishable secure under chosen plaintext attacks (IND-CPA) if no substantial information about the plaintext can be feasibly extracted from the ciphertext. In other words, for a given ciphertext, the distribution of the corresponding plaintext is still random in the view of any PPT adversary.

Definition 2 (*IND-CPA* security) *A fully homomorphic encryption scheme* FHE *is indistinguishable secure under chosen plaintext attacks, if for any PPT adversary $\mathcal{A}$, the following probability holds:*

$$\left| \Pr\left[\mathsf{Expt}^{\text{IND-CPA}}_{\mathsf{FHE}} = 1\right] - \frac{1}{2} \right| = \mathsf{negl}(\lambda),$$

where the security game $\mathsf{Expt}^{\text{IND-CPA}}_{\mathsf{FHE}}$ is defined in Fig. 2.3.

Encoding. Roughly speaking, the encryption algorithm of an LWE-based FHE scheme implicitly consists of an encoding process to convert a message to an element in a special plaintext space, and the decryption also includes a reversed decoding process. In the BFV scheme, a plaintext is defined by an element in a ring $R_t = \mathbb{Z}_t/(x^n + 1)$, whose element is a polynomial having a degree smaller than n and the coefficients come from the field $\mathbb{Z}_t$. t is named as the plaintext modulus, which affects the correctness of an FHE scheme. A bigger t value may allow a larger message space to keep the computation correctness without data-type overflow. However, a too big t value may introduce too much noise during the homomorphic computations. The CKKS scheme extra requires an integer scale factor Δ to convert real inputs to integers. In the following paper, we do not specify the encoding

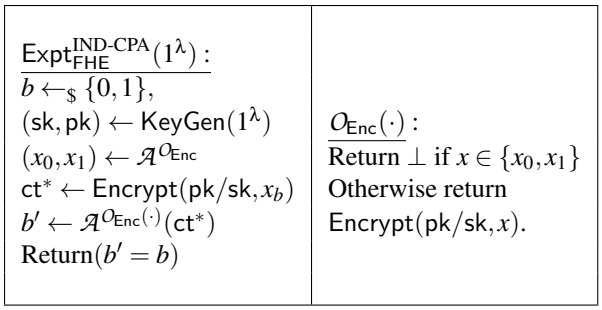

Figure 1: Security game $\mathsf{Expt}^{\text{IND-CPA}}_{\mathsf{HE}}$

algorithm and regard the encoding and decoding processes as the parts of the encryption and decryption, respectively.

Encryption parameters. A BGV-style FHE ciphertext typically contains two polynomials defined in a ring $R_Q = \mathbb{Z}_Q/(x^n+1)$, in which n is the ring dimension and Q is the ciphertext modulo. Selecting a big enough n value would guarantee the security level specified by the security parameter λ [5]. A small Q value cannot support too many computation levels in homomorphic encryption, while a too big ciphertext modulo Q value would introduce more noise. A too big n or Q value would add the storage and computational overhead. Hence, the encryption parameters should be carefully selected to fit a computation task without losing neither security, efficiency, nor correctness.

SIMD batching. One FHE encryption renders a large computational overhead and a large ciphertext storage size. SIMD is a typical trick used in parallel computing, which is widely used in FHE schemes [8,41]. When N messages are embedded into a batched ciphertext, the computational overhead of an encryption/decryption or an evaluation, and the storage usage of the ciphertext, are all divided by N, i.e., an amortization operation. Hence, batching could be regarded as a computation acceleration or a ciphertext compression operation in some sense. Sect. 5 would clearly compare the storage difference between batching and unbatching.

Roughly speaking, when the plaintext modulus t satisfies $t \equiv 1 \mod 2n$, BFV and BGV support an $N = n$ batching at maximum [34]. In CKKS, the maximum batching size is $\frac{n}{2}$. Popular FHE libraries, e.g., PALISADE [38] and SEAL [3], already widely follow this batching technique and highly recommend that the developers should batch their messages as much as possible, although batching does not fit all FHE applications as we discussed in Sect. 1.

2.4 A Straightforward FHE-based Outsourced Computation Protocol

Fig. 2 introduces a straightforward FHE-based protocol used in an outsourced storage and computation scenario. At first, a client sets an FHE system by generating secret and public keys according to a set of encryption parameters. Then, the client encrypts its confidential messages by the FHE scheme, and sends several ciphertexts to the server side, together with the generated public key. The server stores the ciphertexts on behalf of the client. When the client requests the server to conduct some computations on its encrypted data, the client proposes a computation task (generally a circuit Γ). Next, the server computes this task without knowing the message. Finally, the server returns the target ciphertext to the client side, and the client decrypts it to obtain the results.

However, straightforward using an FHE scheme in this scenario faces the storage and computation obstacles. The FHE ciphertext expansion rate is significant. When $n = 8,192$, $t = 65,537$, $\log_2 Q = 180$, encrypting a 16-bit integer message leads to a 386.97 KB-size BFV ciphertext in PALISADE library rendering an expansion rate of around $200\times$. Even when batching is acceptable, the expansion rate of a ciphertext having the maximum batch size is still more than $20\times$.

In addition, the number of ciphertext-wise operations restrains this application from the computation time and introduced noise. Usually, the computation time for a ciphertext-wise multiplication is much higher than an addition or a scalar multiplication. A ciphertext-wise multiplication also introduces a large amount of noise compared with other computations. An addition or a scalar multiplication accumulates much less noise. Since bootstrapping is a heavy computational burden to eliminate noise, a noise optimization is implicit a computation time optimization.

3 The HEAD Protocol

In this section, we propose our generic HEAD protocol for making an FHE-based outsourced computation more practical in the aspects of storage and computation (Sect. 3.1), and then prove its security in a simulation-based model (Sect. 3.2).

3.1 Syntax of the Generic HEAD Protocol

Definition 3 *The generic HEAD protocol is depicted in Fig.3, consisting of six PPT algorithms, namely* HEAD = (KeyGen, Encrypt, Decrypt, Eval, NewEval, Trans).

- The KeyGen, Encrypt, Decrypt, and Eval algorithms follow the original ones in an FHE scheme as we introduced in Sect. 2. In addition, $\{\mathsf{ct}(r_i)\} = \{\mathsf{ct}(r_i) \mid \forall i \in I, r_i \leftarrow_{\$} \mathcal{M}, \mathsf{ct}(r_i) \leftarrow \mathsf{Encrypt}(\mathsf{pk}, r_i)\}$ for an index set I is denoted by $\{\mathsf{ct}(r_i)\} \leftarrow \{\mathsf{Encrypt}(\mathsf{pk}, r_i)\}$, while an SIMD-batched encryption is denoted by $\mathsf{ct}(\{r_i\}) \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \{r_i\})$.
- The transformation algorithm $\mathsf{Trans}(x,r)$ returns a masked ciphertext $\widetilde{x}$ upon input of a message $x \in \mathcal{M}$ and a randomness $r \in \mathcal{M}$. If the inputs are two sets, $\{\widetilde{x_i}\} \leftarrow \mathsf{Trans}(\{x_i\},\{r_i\})$ stands for $\{\widetilde{x_i}\} = \{\widetilde{x_i} \mid \widetilde{x_i} \leftarrow \mathsf{Trans}(x_i, r_i)\}$.

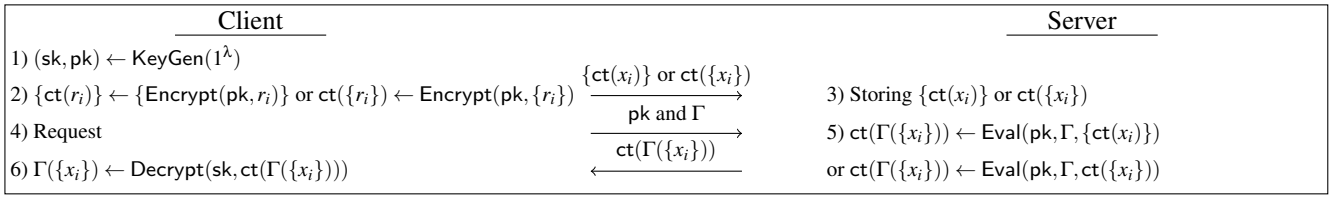

Figure 2: A straightforward FHE-based outsourced computation protocol

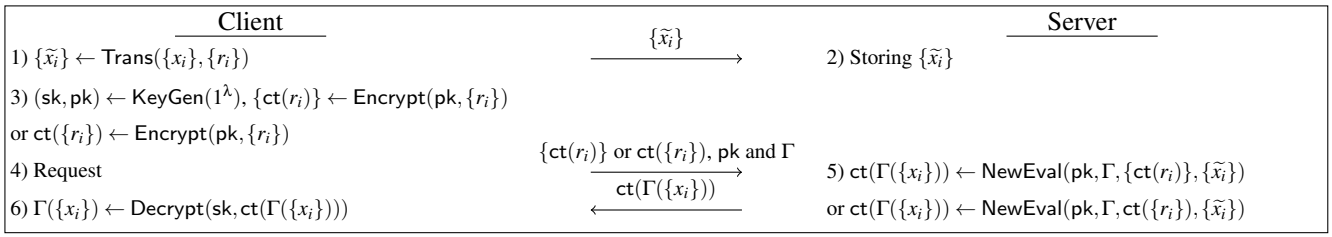

Figure 3: The generic HEAD protocol

- The new evaluation algorithm $\mathsf{NewEval}(\mathsf{pk}, \Gamma, \mathsf{ct}(r), \widetilde{x})$ outputs a ciphertext $\mathsf{ct}(\Gamma(x)) \in \mathcal{CT}$ on input a public key pk, a circuit Γ, an FHE ciphertext $\mathsf{ct}(r)$ and a masked ciphertext $\widetilde{x}$. Still, NewEval could act on unbatched or batched ciphertexts, i.e., $\mathsf{ct}(\Gamma(\{x_i\})) \leftarrow \mathsf{NewEval}(\mathsf{pk}, \Gamma, \{\mathsf{ct}(r_i)\}, \{\widetilde{x_i}\})$ or $\mathsf{ct}(\Gamma(\{x_i\})) \leftarrow \mathsf{NewEval}(\mathsf{pk}, \Gamma, \mathsf{ct}(\{r_i\}), \{\widetilde{x_i}\})$, respectively.

Definition 4 (Correctness) *The generic HEAD protocol is correct if* $\forall x, r \in \mathcal{M}$ *and* $(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$*, the following probability holds:*

$$\Pr\left[\begin{array}{l} \mathsf{Decrypt}(\mathsf{sk}, \mathsf{Encrypt}(\mathsf{pk}, x)) \\ = x \wedge \\ \mathsf{Decrypt}(\mathsf{sk}, \mathsf{Eval}(\mathsf{pk}, \Gamma, \\ \{\mathsf{Encrypt}(\mathsf{pk}, x)\})) \\ = \mathsf{Decrypt}(\mathsf{sk}, \mathsf{NewEval}(\mathsf{pk}, \Gamma, \\ \{\mathsf{Encrypt}(\mathsf{pk}, r), \mathsf{Trans}(x, r)\})) \\ = \Gamma(x_i) \end{array}\right] = 1 - \mathsf{negl}(\lambda)$$

It is obvious that our generic HEAD protocol (Fig. 3) does not change the process of a straightforward FHE-based outsourced computation protocol (Fig. 2), no matter whether an FHE scheme supports SIMD-batching or not. The masked ciphertexts spend much less space achieving compact storage, i.e., the ciphertext expansion rate is 1. Moreover, our protocol offers some computation optimization and higher flexibility. We will describe how we achieve compact storage and efficient computation, and discuss the advantages (including the flexibility) and limitations in Sect. 4.

3.2 Security Analysis

We take the unbatching style to prove the security of the generic HEAD protocol, while the batching case is similar. We consider the security under a semi-honest adversary who has corrupted Server, which means the adversary will honestly execute the protocol with Client but still could not extract any substantial information of Client's messages.

The proof is according to the standard simulation paradigm with the real/ideal model [11, 30]. Informally speaking, for a semi-honest adversary $\mathcal{A}$, we would construct a simulator $\mathcal{S}$ to simulate Client and execute the protocol with $\mathcal{A}$. However, the distribution in the view of $\mathcal{A}$ is indistinguishable from the one in a real execution, which means no leakage about the information of Client's messages.

Theorem 1 *Assume that the underlying fully homomorphic scheme* FHE *is IND-CPA secure, then our generic HEAD protocol is secure in the presence of a semi-honest adversary.*

Proof Let $\mathcal{A}$ be an adversary who has corrupted the Server; we construct a simulator $\mathcal{S}$ to simulate the Client in the HEAD protocol as follows.

- Upon input of a public security parameter λ, a simulator $\mathcal{S}$ invokes $\mathsf{sk},\mathsf{pk} \leftarrow \mathsf{KeyGen}(1^\lambda)$ and receives back $(\mathsf{sk},\mathsf{pk})$.
- $\mathcal{S}$ invokes $\mathcal{A}$ upon input of pk.
- $\mathcal{S}$ randomly chooses $\{x_i, r_i \leftarrow_\$ \mathcal{M},\ \forall i \in I\}$, generates $\{\widetilde{x_i}\} \leftarrow \mathsf{Trans}(\{x_i\}, \{r_i\})$, and computes $\{\mathsf{ct}(r_i)\} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \{r_i\})$.
- $\mathcal{S}$ ramdomly generates a circuit Γ and invokes $\mathcal{A}$ upon input of $(\Gamma, \{\mathsf{ct}(r_i)\}, \{\widetilde{x_i}\})$.

- $\mathcal{S}$ receives $\mathsf{ct}(\Gamma(\{x_i\}))$ and outputs the plaintext $\Gamma(\{x_i\}) \leftarrow \mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}(\Gamma(\{x_i\})))$.

We prove that the distribution of $\mathcal{A}$'s view in the simulation is indistinguishable from the distribution in a real protocol execution. The main difference between the simulation by $\mathcal{A}$ and a real execution with an honest Client is the way that $\widetilde{x_i}$ is generated: the Client internally decides its own message x'_i in $\mathcal{M}$, chooses random r'_i and computes $\widetilde{x'_i} \leftarrow \mathsf{Trans}(x'_i, r'_i)$; whereas $\mathcal{S}$ chooses random x_i and computes $\widetilde{x_i} \leftarrow \mathsf{Trans}(x_i, r_i)$. Since the underlying fully homormophic encyption scheme FHE is IND-CPA secure, and r_i, r'_i are chosen randomly, the distributions over the outputs of $\mathsf{Trans}(x_i, r_i)$ and $\mathsf{Trans}(x'_i, r'_i)$ are indistinguishable in $\mathcal{A}$'s view on receiving $\mathsf{ct}(r_i)$. Otherwise, we could construct a distinguisher D for the IND-CPA security experiment $\mathsf{Expt}^{\text{IND-CCA}}_{\mathsf{FHE}}$ naturally. Thus, we conclude that the distribution of the semi-honest adversary $\mathcal{A}$'s view is indistinguishable from the distribution in a real execution, and finish the proof. ■

4 Various HEAD instantiations

In this section, we instantiate our generic HEAD protocol rendering three protocols, i.e., a modulo-subtraction-masking protocol, a modulo-division-masking protocol, and an XOR-masking protocol. All instantiations achieve compact storage, i.e., decreasing the masked ciphertext expansion rate to 1, and adaptively optimize some computation types, in the FHE-based outsourced computation scenario (Sect. 2.4). Our NewEval function includes two parts, i.e., an unmasking operation and an evaluation. Note that the evaluation may point to many computation steps. Our optimization would take effects on parts of steps, which would be described in corresponding subsections and figures. The concrete computation time and noise reduction by the protocols would be exhibited in Sect. 5.

4.1 Modulo Subtraction Masking

Fig. 4 instantiates Trans by modulo-subtraction-masking. In addition to a ciphertext expansion rate of 1, the computation optimization lies in a reduction from arbitrary times of ciphertext-wise additions or a scalar multiplication to a single ciphertext-plaintext addition, significantly reducing the computation time or noise, respectively. We describe this protocol for an integer input vector, which is suitable for an integer FHE scheme such as BFV [21] or BGV [9].

Storage optimization for an integer number FHE scheme. We consider an application based on a message space $\mathcal{M}$, whose length is a factor of the word length (e.g., 16 bits) and is slightly smaller than $\mathbb{Z}_t$. For Trans in step 1 in Fig. 4, a client firstly masks $\{x_i \in \mathcal{M}\}$ by element-wise modulo subtracting an integer pseudorandom vector $\{r_i\}$ (generated from $\mathsf{PRF}(\mathsf{key}, i)$), i.e., $\mathsf{Trans}(\{x_i\}, \{r_i\}) = \{x_i - r_i \bmod t,\ \forall i \in I\}$. Since the length of $\mathcal{M}$ is a factor of a word length, the size of the masking result $\{\widetilde{x_i}\}$ could be stored in the same size as the one of the original messages, i.e., $|\{\widetilde{x_i}\}| = |\{x_i\}|$ and the ciphertext expansion rate is 1. The client transfers the masked set $\{\widetilde{x_i}\}$ to the server side. Then, the client stores the small-size PRF key for a long time as needed, until a computation request is raised.

Next, we show how this modulo-subtraction-masking could be unmasked using the FHE homomorphic properties, i.e., step 5.1 in Fig. 4. When the client wants to launch a computation on its ciphertexts, the client re-generates the pseudorandom vector $\{r_i\}$ from the PRF using the same key, and generates the FHE scheme (e.g., BFV or BGV) secret and public keys. The FHE scheme should be configured by the same plaintext modulus t as the one in the masking process, while other configuarations could be flexible according to the proposed computation circuit Γ. The masking set is encrypted into $\mathsf{ct}(\{r_i\})$ in an SIMD-batching style, and $\mathsf{ct}(\{r_i\})$ would be sent to the server along with the public key and Γ. After receiving the request, the server picks up the ciphertext $\{\widetilde{x_i}\}$ from its storage. The server would obtain $\mathsf{ct}(\{x_i\})$ by computing $\mathsf{ct}(\{\widetilde{x_i}\}) \leftarrow \{\widetilde{x_i}\} + \mathsf{ct}(\{r_i\}) = \mathsf{ct}(\{\widetilde{x_i}\} + \{r_i\})$, in which $\{\widetilde{x_i}\} + \mathsf{ct}(\{r_i\})$ is an SIMD-batched addition between a batched plaintext and a batched ciphertext, and $\{\widetilde{x_i}\} + \{r_i\}$ stands for an element-wise addition for $\{\widetilde{x_i} + r_i,\ \forall i \in I\}$.

The server then computes the task according to the circuit description Γ, and returns a result ciphertext $\mathsf{ct}(\Gamma(\{x_i\}))$ to the client. Finally, the client decrypts it to have $\Gamma(\{x_i\})$.

Computation optimization for one or more ciphertext-wise additions and a scalar multiplication. Since this modulo-subtraction-masking is linear, the masked message still keeps the linear homomorphic properties, which could move a linear FHE ciphertext evaluation to the masking side. The masking side operations are much easier than the ones in the FHE ciphertext side. In step 2 of Fig. 4, the client may preprocess $\mathsf{ct}(\sum_{i=1}^{k} r_i)$ or $\mathsf{ct}(c \cdot r_i)$ if the computation task Γ requires a large number of additions or a scalar multiplication.

Step 5.2 in Fig. 4 reduces the sum of k FHE ciphertexts $\sum_{i=1}^{k} \mathsf{ct}(\mathsf{x}_i)$ to a single FHE ciphertext-plaintext addition $\sum_{i=1}^{k} \widetilde{x_i} + \mathsf{ct}(\sum_{i=1}^{k} r_i)$. This is a significant optimization since the computational overhead for the sum of k masked ciphertexts or k pseudorandoms is much small. Furthermore, when the inputs are batched in one FHE ciphertext, the straightforward FHE computation for the sum of k ciphertexts $(\sum_{i=1}^{k} \mathsf{ct}(x_i))$ should be accompanied with needed rotations or evaluations at indexes, so that our technique would further reduce the computation time. This transformation also decreases a little introduced noise.

The optimization also works for an FHE scalar multiplication $c \cdot \mathsf{ct}(x_i)$ by computing the scaling on the masking side and the pseudorandom side, i.e., $c \cdot \widetilde{x_i}$ and $c \cdot r_i$, and then computing a plaintext-ciphertext addition $c \cdot \widetilde{x_i} + \mathsf{ct}(c \cdot r_i)$. Although this optimization decreases a little computation time, the reduced noise volume is significant.

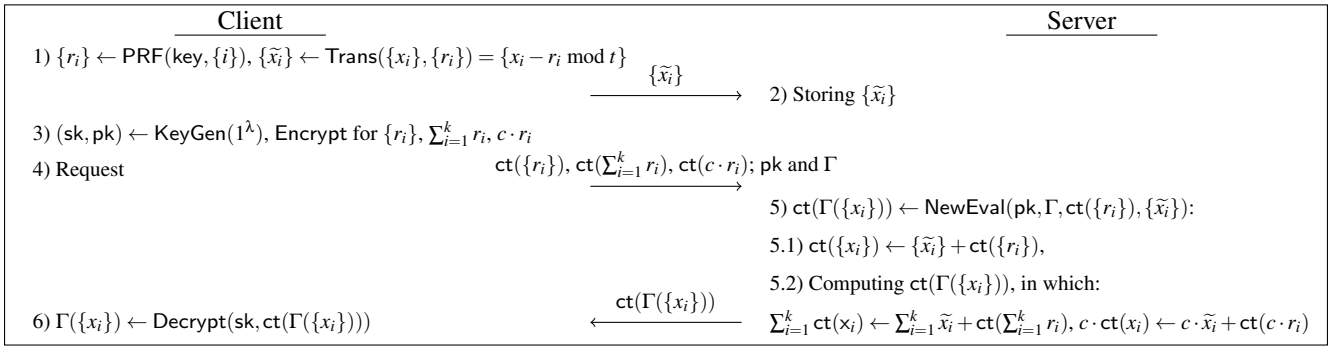

Figure 4: The modulo-subtraction-masking HEAD protocol for an integer number FHE scheme

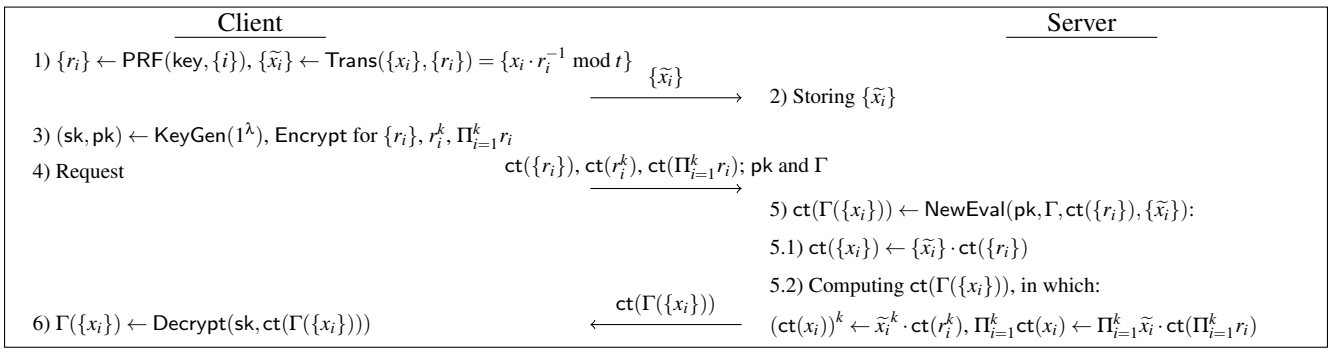

Figure 5: The modulo-division-masking HEAD protocol for an integer number FHE scheme

4.2 Modulo Division Masking

Fig. 5 introduces the modulo-division-masking HEAD protocol, i.e., instantiating Trans by a modulo division for integer number inputs. Still, this protocol not only achieves compact storage, but also decreases the FHE ciphertext multiplication consumptions at the server side. The computation optimization reduces arbitrary times of ciphertext multiplications to a single scalar multiplication, significantly decreasing both the computation time and noise.

Storage optimization for a non-zero integer number FHE scheme. This protocol has similar requirements for $\mathcal{M}$ and t to the ones in the modulo-subtraction-masking HEAD protocol (Sect. 4.1). For non-zero integer inputs $\{x_i\}$, Trans invokes $\{x_i \cdot r_i^{-1} \bmod t,\ \forall i \in I\}$, in which non-zero $\{r_i\}$ comes from a PRF. The unmasking operation inside NewEval only consumes one SIMD-batched scalar multiplication to unmask the maskings, i.e., $\mathsf{ct}(\{x_i\}) \leftarrow \{\widetilde{x_i}\} \cdot \mathsf{ct}(\{r_i\})$.

Computation optimization for one or more ciphertext-wise multiplications. Step 5.2 in Fig. 5 introduces the modulo-division-masking computation optimization. If a computation request Γ includes the k power of an FHE ciphertext, the client in HEAD would prepare $\{r_i^k\}$ by computing the k power of each element r_i, then send the ciphertext $\mathsf{ct}(\{r_i^k\})$ in the request. The server computes $\{\widetilde{x}^k\}$ and then only consumes one scalar multiplication to compute $(\mathsf{ct}(x_i))^k$.

If Γ requires a k-degree cross term from k variables, i.e., $\Pi_{i=1}^k x_i$. The straightforward FHE usage requires rotations or evaluations at different indexes to achieve the product of k ciphertexts $\Pi_{i=1}^k \mathsf{ct}(x_i)$. Our optimization also works in this case, by firstly computing $\Pi_{i=1}^k \widetilde{x_i}$ and $\Pi_{i=1}^k r_i$ then computing one scalar multiplication of $\Pi_{i=1}^k \widetilde{x_i} \cdot \mathsf{ct}(\Pi_{i=1}^k r_i)$.

These optimizations decrease both a large amount of computation time and noise, compared with the heavy homomorphic operations in the straightforward FHE usage.

4.3 XOR Masking

The XOR-masking HEAD protocols (in Fig. 6 and Fig. 7) work for a real or binary number input. Both two protocols achieve compact storage.

Storage optimization for a real number FHE scheme. We consider a fixed-point real number x occupying m bits in Fig. 6, and x^j stands for the j-th bit for $j \in [0, m-1]$. Specifically, the first m_{frac} bits represent the fraction part of this message, i.e., $x^0, \cdots, x^{m_{\mathsf{frac}}-1}$, while the remaining $m - m_{\mathsf{frac}}$ bits indicate the integer part, i.e., $x^{m_{\mathsf{frac}}}, \cdots, x^{m-1}$. Hence, the binary representation of x satisfies $x = \sum_{j=0}^{m_{\mathsf{frac}}-1} x^j \cdot 2^{j-m_{\mathsf{frac}}} +$

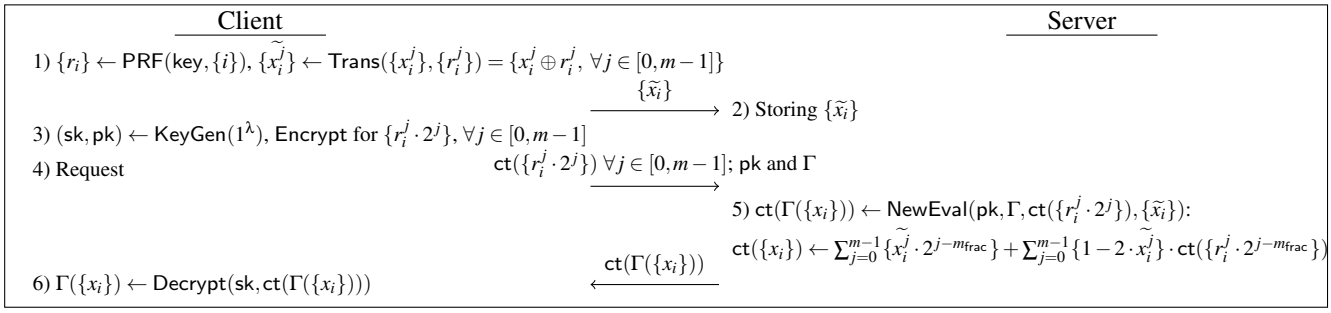

Figure 6: The XOR-masking HEAD protocol for a real number FHE scheme

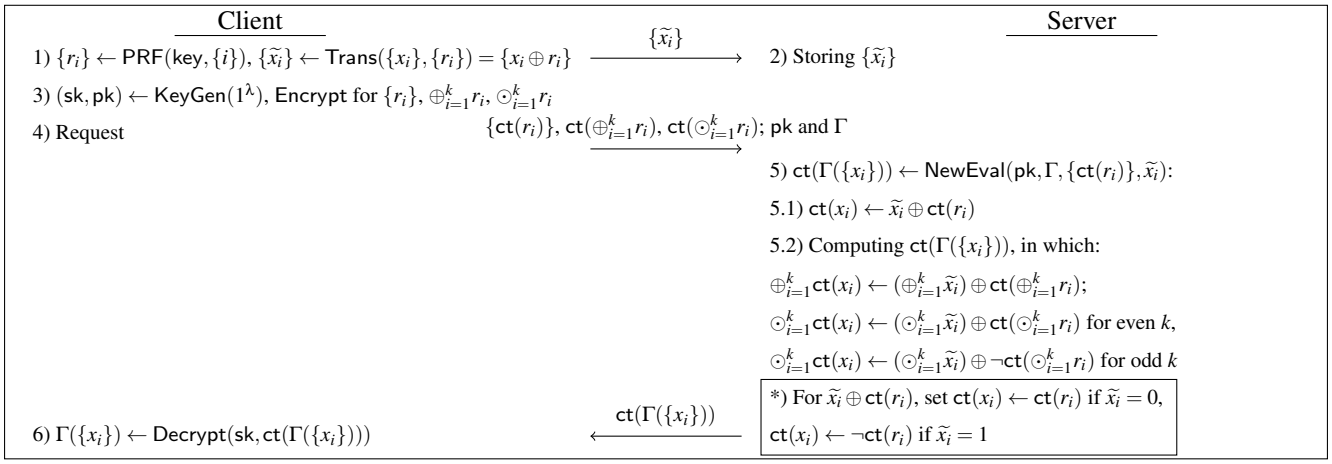

Figure 7: The XOR-masking HEAD protocol for a binary FHE scheme

$\sum_{j=m_{\mathsf{frac}}}^{m-1} x^j \cdot 2^{j-m_{\mathsf{frac}}} = \sum_{j=0}^{m-1} x^j \cdot 2^{j-m_{\mathsf{frac}}}$.

The Trans operation masks each bit of the input set $\{x_i^j\}$ by an XOR operation with another random bit r_i generated from a PRF, i.e., $\{\widetilde{x_i^j} \leftarrow x_i^j \oplus r_i^j,\ \forall j \in [0, m-1],\ \forall i \in I\}$.

Since a real number FHE scheme like CKKS supports SIMD-batching, NewEval may execute the unmasking operations in a batching style. For each bit of each element x_i^j, the binary XOR could be converted to a decimal relation, i.e., $x_i^j = \widetilde{x_i^j} \oplus r_i^j = \widetilde{x_i^j} + r_i^j - 2 \cdot \widetilde{x_i^j} \cdot r_i^j$. For an m-bit fixed-point decimal real number, we have

$$x_i = \sum_{j=0}^{m-1} \left(\widetilde{x_i^j} \cdot 2^{j-m_{\mathsf{frac}}} + r_i^j \cdot 2^{j-m_{\mathsf{frac}}} - 2 \cdot \widetilde{x_i^j} \cdot r_i^j \cdot 2^{j-m_{\mathsf{frac}}} \right)$$
$$= \sum_{j=0}^{m-1} \left(\widetilde{x_i^j} \cdot 2^{j-m_{\mathsf{frac}}} + (1 - 2 \cdot \widetilde{x_i^j}) \cdot r_i^j \cdot 2^{j-m_{\mathsf{frac}}} \right)$$

Hence, the relation in SIMD-batched ciphertexts keeps,

$$\mathsf{ct}(x_i) = \sum_{j=0}^{m-1} \left(\widetilde{x_i^j} \cdot 2^{j-m_{\mathsf{frac}}} + (1 - 2 \cdot \widetilde{x_i^j}) \cdot \mathsf{ct}(r_i^j \cdot 2^{j-m_{\mathsf{frac}}}) \right),$$

$$\mathsf{ct}(\{x_i\}) = \sum_{j=0}^{m-1} \left(\{\widetilde{x_i^j} \cdot 2^{j-m_{\mathsf{frac}}}\} + \{1 - 2 \cdot \widetilde{x_i^j}\} \cdot \mathsf{ct}(\{r_i^j \cdot 2^{j-m_{\mathsf{frac}}}\}) \right)$$
$$= \sum_{j=0}^{m-1} \{\widetilde{x_i^j} \cdot 2^{j-m_{\mathsf{frac}}}\} + \sum_{j=0}^{m-1} \{1 - 2 \cdot \widetilde{x_i^j}\} \cdot \mathsf{ct}(\{r_i^j \cdot 2^{j-m_{\mathsf{frac}}}\}).$$

Note that $\{\widetilde{x_i^j} \cdot 2^{j-m_{\mathsf{frac}}}\}$ and $\sum_{j=0}^{m-1} \{\widetilde{x_i^j} \cdot 2^{j-m_{\mathsf{frac}}}\}$ are element-wise multiplication and sum operations for all elements in the sets, respectively. $\{1 - 2 \cdot \widetilde{x_i^j}\} \cdot \mathsf{ct}(\{r_i^j \cdot 2^{j-m_{\mathsf{frac}}}\})$ and $\sum_{j=0}^{m-1} \{1 - 2 \cdot \widetilde{x_i^j}\} \cdot \mathsf{ct}(\{r_i^j \cdot 2^{j-m_{\mathsf{frac}}}\})$ are SIMD-batched scalar multiplication and sum operations, respectively.

Before a request, the client prepares $\mathsf{ct}(\{r_i^j \cdot 2^{j-m_{\mathsf{frac}}}\})$ for $\forall j \in [0, m-1],\ \forall i \in I$, then transfers these m ciphertexts to the server side. After unmasking and computations, the CKKS ciphertext $\mathsf{ct}(\Gamma(\{x_i\}))$ is transferred to the client.

Storage optimization for a binary input FHE scheme. The XOR-masking HEAD protocol (Fig. 7) for a binary FHE scheme takes the unbatching style since a GSW-style FHE scheme does not support SIMD-batching. The message space is $\mathcal{M} = \{0, 1\}$. For masking an binary vector $\{x_i\}$, the client runs bit-wise XOR by another binary pseudorandom vector

$\{r_i\}$ in Trans, rendering the same size vector $\{\widetilde{x_i}\}$, which prevents the ciphertext expansion. For unmasking, this XOR HEAD protocol requires a communication overhead to transfer the unbatched FHE ciphertexts, $\{\mathsf{ct}(r_i)\}$. Then, the server independently computes $\widetilde{x_i} \oplus \mathsf{ct}(r_i)$ to recover $\mathsf{ct}(x_i)$.

Computation optimization for one or more ciphertext-wise XOR or XNOR operations for a binary FHE scheme. Step 5.2 in Fig. 7 introduces our computation optimization for this protocol. If a computation request Γ requires the XOR or XNOR sum of k ciphertexts (i.e., $\oplus_{i=1}^{k}\mathsf{ct}(x_i)$ or $\odot_{i=1}^{k}\mathsf{ct}(x_i)$), the client would prepare and send the ciphertext $\mathsf{ct}(\oplus_{i=1}^{k} r_i)$ or $\mathsf{ct}(\odot_{i=1}^{k} r_i)$ along with the request, respectively. Since the XOR operation is transitive, we have $\oplus_{i=1}^{k}\mathsf{ct}(x_i) \leftarrow (\oplus_{i=1}^{k}\widetilde{x_i}) \oplus \mathsf{ct}(\oplus_{i=1}^{k} r_i)$, which is a single ciphertext-wise XOR operation. One XOR operation based on two FHE ciphertexts is very slow as the exhibition in Sect. 5. Hence, our optimization is significant since we reduce the heavy burden to compute $k-1$ times of XOR for $\oplus_{i=1}^{k}\mathsf{ct}(x_i)$ to a single XOR.

The optimization also works for XNOR. From $a \odot b = \neg(a \oplus b)$ and $a \odot b \odot c = a \oplus b \oplus c$, we could induce that $\odot_{i=1}^{k} r_i$ equals to $\oplus_{i=1}^{k} r_i$ or $\neg(\oplus_{i=1}^{k} r_i)$ if k is odd or even, respectively. Hence, the XNOR sum of k ciphertexts $\odot_{i=1}^{k}\mathsf{ct}(x_i)$ could be optimized to $(\odot_{i=1}^{k}\widetilde{x_i}) \oplus \mathsf{ct}(\odot_{i=1}^{k} r_i)$ or $(\odot_{i=1}^{k}\widetilde{x_i}) \oplus \neg\mathsf{ct}(\odot_{i=1}^{k} r_i)$ for an even k or odd k, respectively. Similarly, the ciphertext-wise XNOR operation is heavy, so that our optimization is also significant.

Supporting a symmetric binary FHE scheme. From our implementation experience, two famous FHE open-source libraries PALISADE [38] and TFHE [16] only support the symmetric version of FHE schemes, which does not support a plaintext-ciphertext XOR or an encryption before a ciphertext-wise XOR at the server side in an FHE-based outsourced computation scenario.

In order to apply this XOR-masking protocol in practice, we design another method at the server side (step * in Fig. 7). For computing $(\widetilde{x_i}) \oplus \mathsf{ct}(r_i)$, the operation is to reverse $\mathsf{ct}(r_i)$ (a NOT ciphertext operation) in essence, according to the bit value of the $\widetilde{x_i}$. Since $x_i \leftarrow \widetilde{x_i} \oplus r_i$, $x_i \leftarrow 0 \oplus r_i = r_i$ if $\widetilde{x_i} = 0$ and $x_i \leftarrow 1 \oplus r_i = \neg r_i$ if $\widetilde{x_i} = 1$. Hence, the server could set $\mathsf{ct}(x_i) \leftarrow \mathsf{ct}(r_i)$ if $\widetilde{x_i} = 0$, or $\mathsf{ct}(x_i) \leftarrow \neg\mathsf{ct}(r_i)$ if $\widetilde{x_i} = 1$. A NOT ciphertext operation takes a much less time (the level of microseconds) than other ciphertext-wise Boolean operations (the level from hundreds of milliseconds or seconds).

4.4 Discussion

Compact storage: the masked ciphertext expansion rate of 1. Similar to Gentry et al.'s work [27, 28], a server in our HEAD protocols stores compact ciphertexts in an FHE-based outsourced computation scenario. When the client requests a computation, the AES ciphertexts in Gentry et al.'s work [27, 28] and the masked ciphertexts are transferred to high-expansion FHE ciphertexts. Both Gentry et al.'s work [27, 28] and ours do not change a straightforward FHE ciphertext expansion rate. However, our protocol is more generic and extra offers computation optimizations than Gentry et al.'s work [27, 28], by paying a little communication overhead.

We regard a used FHE scheme as a black box in our protocols, which offers us more genericity without relying on a specific FHE scheme. Gentry et al.'s work [27, 28] only could store the AES ciphertext and transfer it to a BGV FHE ciphertext. Our protocols support various masking techniques and FHE schemes depending on the outsourced computation data type. In addition, a masked ciphertext could be transformed faster and could keep or even optimize some computation, compared with an AES ciphertext. While the computation of an AES ciphertext is limited in the field $GF(2^d)$.

A transformed FHE ciphertext in our protocol or Gentry et al.'s work [27, 28] may still occupy some memory space when loading the FHE ciphertext for computation, which would be one of our future works.

Adaptively supporting different computation types. The three HEAD protocols optimize different computations depending on the masking technique used in a storage phase. Since the masked ciphertext expansion rate is 1 in all the three instantiations, a client may outsource long-term storage for all the three types (modulo-subsection, modulo-division, and XOR) of masked messages, using three different pseudorandom keys, without occupying too much space. No matter for what kind of a computation task, the corresponding maskings are picked up and used in the following computation.

The flexibility of HEAD protocols. It is worth noticing that the server may store the masked ciphertexts during the lifecycle of the outsourced storage and computation application. Our protocol is not of one-time use. After one-time outsourced storage, the client may propose multiple computation requests on its outsourced ciphertexts without changing the pseudorandom maskings.

A client may upload its I_1 masked messages at the beginning, while each following requested computation may only require I_2 messages and $I_2 << I_1$, i.e., a shorter pseudorandom array as needed, which would further save the communication overhead for the corresponding FHE ciphertext(s). Also, the indexes included in I_2 are not limited. The retrieved messages could be transformed to one or more FHE ciphertexts as required if the selected FHE scheme supports SIMD-batching, without the need to carry out unnecessary rotations.

What's more, the encryption parameters of a transformed FHE ciphertext could be configured just before a computation task is proposed. The client configures an FHE scheme adaptively according to an outsourced computation task.

The extra costs of HEAD. As we depicted in Fig. 2 and Fig. 3, the HEAD protocols do not change the procedure of a straightforward FHE-based outsourced computation protocol too much. In exchange for compact storage and computation optimization, the HEAD protocols may sacrifice some communication overhead to transfer some FHE ciphertexts

of (the variants of) the pseudorandom maskings. Since, BFV and BGV support SIMD-batching, the ModDiv and ModSub communication overheads are small, only transferring a batched ciphertext. We also utilize the SIMD-batching supporting for CKKS, so that the XOR-masking protocol for CKKS could transfer m batched ciphertexts when the input real numbers are m-bit fixed-point numbers. Since FHEW (or other GSW-style FHE schemes) does not support batching, both our XOR-masking protocol and a straightforward FHEW-based outsourced computation protocol would transfer unbatched FHEW ciphertexts.

For the unmasking operations, the computational overhead is small or reasonable in different HEAD protocols, as we discussed in the previous subsections.

The trade-off between compact storage and computation optimization for CKKS. This paper mainly describes our XOR-masking HEAD protocol for CKKS (Fig. 6), achieving the extremely compact ciphertext storage without computation optimization. However, the ModSub protocol also supports CKKS, similarly optimizes additions and scalar multiplications, with a small ciphertext expansion.

For a real number x, a scale-round-modulo operation renders $(\lfloor x \cdot \Delta \rceil \mod Q) \in \mathcal{M}$. The scale factor Δ would be the same as the one used in the following CKKS scheme. Implementing Trans by $\{\widetilde{x_i} = \lfloor x_i \cdot \Delta \rceil - r_i \mod Q\}$ and selecting $r_i \leftarrow_{\$} \mathbb{Z}_Q$ render the masking size expansion, since usually the value of Q is not small. Considering a double-type input x and $\log_2 Q = 150$, the expansion rate is around $\frac{150}{64} \approx 3$.

For the unmasking operation inside $\mathsf{NewEval}$, each element in $\{\widetilde{x_i}\}$ would be multiplied by the scale factor inverse, i.e., $\{\widetilde{x_i} \cdot \Delta^{-1}\}$. The client prepares $\mathsf{ct}(\{r_i \cdot \Delta^{-1}\})$ and then the server computes $\{\widetilde{x_i} \cdot \Delta^{-1}\} + \mathsf{ct}(\{r_i \cdot \Delta^{-1}\})$, i.e., an SIMD-batched plaintext-ciphertext addition. The masked messages in this way still keep the similar computation optimizations as the ones in the integer case, i.e., step 5.2 in Fig. 4.

The value of a plaintext modulus t. As we introduced in Sect. 2, an FHE evaluation result is t-modulo after several computations. Different applications have different requirements of the t value. However, no matter what the t value is, our ModSub and ModDiv protocols keep compact storage in an FHE-based outsourced computation scenario.

The open question whether a real input could be division masked to enhance the ciphertext multiplication strength. According to our security proof for the HEAD protocol in Sect. 3, a masking method is secure when a mask and the resulting masked message are indistinguishable. The ratio $\widetilde{x} = \frac{x}{r}$ would be too small or too large if the double type variables x and r come from the same space, which is beyond the precision of an FHE scheme like CKKS. Hence, how to use the HEAD protocol to mask a real number input in a division form would be one of our future works.

5 Performance Evaluation

This section evaluates the performance of our three HEAD protocols in storage and computation aspects. We also exhibit the straightforward usage of FHE schemes without our protocols as a baseline. We mainly evaluate the performance using the PALISADE library [38] [5], while also obtaining the introduced noise volume from the SEAL library [3]. The testing environment is based on Intel Xeon Gold 6266C CPU @ 3.00GHz and 64 GB memory, with Ubuntu 18.04.2 operation system and GCC 7.5.0 compiler.

5.1 ModSub for BFV and BGV

To support the modulo-subtraction-masking (ModSub) protocol (Fig. 4), we first select the encryption parameters for BFV and BGV schemes to ensure the FHE security and usability. Since two straightforward FHE cases, i.e, an 8,192-size batch and 8,192 unbatched ciphertexts, are executed to exhibit the ciphertext expansion, the ring dimension n is set to $8,192$ to support that batch size. The length of ciphertext modulus Q is $\log_2 Q = 180$ for some linear computations. We use an $\mathsf{unsigned\ integer}$ variable to load an input message from $\{0,1\}^{16}$, so that the ciphertext expansion rate is calculated by $\frac{\mathsf{ct_{size}}}{2\mathrm{B}}$. The plaintext modulus $t = 65,537$ ensures the plaintext would not overflow. This setting is deployed in Table 2 and Table 3 (denoted by Setting I).

Table 2 shows the ciphertext size in the BFV and BGV schemes without our protocols. One unbatched BFV ciphertext takes 386.97 KB, and the ciphertext expansion rate is around $\frac{396{,}261\ \mathrm{B}}{2\ \mathrm{B}} \approx 198,130.5$. Then, 8,192 unbatched BFV ciphertexts occupy a considerable space of 3.02 GB. An 8,192-batched BFV ciphertext achieves an amortized expansion rate of $\frac{396{,}261\ \mathrm{B}}{2\ \mathrm{B} \cdot 8{,}192} \approx 24.2$. The expansion rates keep the same order of magnitude for BGV. Note that a batched ciphertext may not support all applications as we discussed previously.

Our ModSub protocol keeps a ciphertext expansion rate of 1, i.e., $\frac{16\ \mathrm{KB}}{2\ \mathrm{B} \cdot 8{,}192} = 1$, supporting the BFV and BGV schemes in the tests. A masked message vector can be "batched" store, while the usage of each vector variable is flexible. When the batch size is 8,192, Trans (Step 1 in Fig. 4), i.e., generating $\{r_i\}$ and computing $\{x_i - r_i \mod t\}$, costs 510 μs in total. Then, it spends around 279 μs for a later re-generation of $\{r_i\}$ from the PRF. If the computation requires all stored messages, one batched unmasking computation for all masked ciphertexts costs only 9.4 ms for necessary encodings combined with one plaintext-ciphertext addition (Step 5.1 in Fig. 4). For BGV, the unmasking time takes a smaller value.

Table 3 exhibits the computation optimization by our ModSub protocol, which reduces one or more ($k-1$ times of) ciphertext-wise additions ($\mathsf{ct} + \mathsf{ct}$) or a scalar multiplication ($\mathsf{ct} \cdot \mathsf{pt}$) to a single plaintext-ciphertext addition ($\mathsf{ct} + \mathsf{pt}$). We

[5] Currently, the PALISADE library supports most FHE schemes.

Table 2: Storage optimization for ModSub HEAD (Ciphertext expansion rate abbreviated as CER)

Scheme	Straightforward FHE (Unbatch)		Straightforward FHE (Batch)		ModSub HEAD			
	$\mathsf{ct}_{\mathsf{size}}$	CER	$\mathsf{ct}_{\mathsf{size}}$	CER	$\mathsf{ct}_{\mathsf{size}}$	CER	$\mathsf{Trans}_{\mathsf{time}}$	$\mathsf{Unmask}_{\mathsf{time}}$
BFV	3,246,170,112 B (3.02 GB)	198,130.5	396,261B (386.97 KB)	24.2	16 KB	1	510 μs	9.4 ms
BGV	4,320,976,896 B (4.02 GB)	363,731.5	527,463 (515.10 KB)	32.2	16 KB	1		6.3 ms

Table 3: Computation optimization for ModSub and ModDiv HEAD

Scheme	Setting I			Setting II		
	Task	Straightforward FHE time	ModSub time	Task	Straightforward FHE time	ModDiv time
BFV	$\mathsf{ct}+\mathsf{pt}$	6.3 ms	6.3 ms	$(k-1)\cdot(\mathsf{ct}\times\mathsf{ct})$	$(k-1)\cdot$131.5 ms	9.5 ms
	$(k-1)\cdot(\mathsf{ct}+\mathsf{ct})$	$(k-1)\cdot$6.3 ms		$(k-1)\cdot(\mathsf{ct}\times\mathsf{ct})^*$	$(k-1)\cdot$132.1 ms	
	$\mathsf{ct}\cdot\mathsf{pt}$	6.4 ms		$(\mathsf{ct}(x))^8$	398.1 ms	
	$\sum_{i=1}^{8}\mathsf{ct}(x_i)$	336.3 ms		$\Pi_{i=1}^{8}\mathsf{ct}(x_i)$	1,219.4 ms	
BGV	$\mathsf{ct}+\mathsf{pt}$	3.2 ms	3.2 ms	$(k-1)\cdot(\mathsf{ct}\times\mathsf{ct})$	$(k-1)\cdot$93.8 ms	9.8 ms
	$(k-1)\cdot(\mathsf{ct}+\mathsf{ct})$	$(k-1)\cdot$6.4 ms		$(k-1)\cdot(\mathsf{ct}\times\mathsf{ct})^*$	$(k-1)\cdot$100.8 ms	
	$\mathsf{ct}\cdot\mathsf{pt}$	9.7 ms		$(\mathsf{ct}(x))^8$	357.8 ms	
	$\sum_{i=1}^{8}\mathsf{ct}(x_i)$	423.6 ms		$\Pi_{i=1}^{8}\mathsf{ct}(x_i)$	1,311.4 ms	
	$\sum_{i=1}^{8}\mathsf{ct}(x_i)^*$	183.8 ms		$\Pi_{i=1}^{8}\mathsf{ct}(x_i)^*$	585.0 ms	

Table 4: Storage optimization for XOR HEAD (Ciphertext expansion rate abbreviated as CER)

Scheme	Straightforward FHE (Unbatch)		Straightforward FHE (Batch)		XOR-masking HEAD			
	$\mathsf{ct}_{\mathsf{size}}$	CER	$\mathsf{ct}_{\mathsf{size}}$	CER	$\mathsf{ct}_{\mathsf{size}}$	CER	$\mathsf{Trans}_{\mathsf{time}}$	$\mathsf{Unmask}_{\mathsf{time}}$
CKKS	8,615,895,040 B (8.02 GB)	525,872.5	1,051,745 B (1,027.09 KB)	64.2	16 KB	1	253 μs	289.4 ms
FHEW	33,923,072 B (32.35 MB)	33,128	N.A.	N.A.	8,192×1 bit	1		8,192×2 μs

take a sum case to reflect the promising computation optimization. When computing the sum of $k=8$ ciphertexts, it is obvious that equally computing $\mathsf{pt}(8)\cdot\mathsf{ct}(x)$ is the fastest way if the eight ciphertexts are identical, consuming a scalar multiplication for 6.4 ms. When the accumulated items are disparsed in one ciphertext batch, there are extra 7 operations for "evaluation at index" and 7 additions, leading to around 336.3ms to compute $\sum_{i=1}^{8}\mathsf{ct}(x_i)$ in total. For both $\mathsf{pt}(8)\cdot\mathsf{ct}(x)$ and $\sum_{i=1}^{8}\mathsf{ct}(x_i)$, ModSub could outputs the corresponding results for 6.3ms. No matter how many numbers of additions ($k-1$ additions takes $(k-1)\cdot$6.3ms) could be reduce to a single plaintext-ciphertext addition (taking 6.3ms).

For BGV, the PALISADE library release 1.11.5 extra offers the rotation functionality. The sum could be accomplished by rotations if $x_1,\cdots,x_8$ are arranged in order in a batch [6], which is denoted by $\sum_{i=1}^{8}\mathsf{ct}(x_i)^*$ in Table 3. ModSub takes 3.2 ms for the sum of eight ciphertexts, faster than both an evaluation-at-index sum and a rotation sum if the batched items are arranged in order or not respectively.

Table 5 exhibits our BFV noise optimization from the SEAL library [3]. Since a ciphertext-wise addition introduces not much noise to the ciphertext, the optimization is slight for several bits when the sum size k is not large [7]. However, the noise of a scalar multiplication is 21 bits in our setting. It makes our optimization obvious since a plaintext-ciphertext addition introduces tiny noise.

5.2 ModDiv for BFV and BGV

In order to support a deeper circuit in the tests, we set the encyption parameters by $n=8,192, t=65,537, \log_2 Q=180$ for BFV and $n=16,384, t=65,537, \log_2 Q=225$ for BGV, respectively, which is the Setting II in Table 3.

The modulo-division-masking (ModDiv) protocol also offers a ciphertext expansion rate of 1. The difference lies in the masking and unmasking overhead. The masking operations

[6] At first, left-rotate the batch once, then compute an addition to get $\mathsf{ct}(x_1)+\mathsf{ct}(x_2)$. Next, left-rotate the result by 2 slots, and an addition renders $\mathsf{ct}(x_1)+\cdots+\mathsf{ct}(x_4)$. Finally, a 4-slot left-rotation and an addition lead to $\sum_{i=1}^{8}\mathsf{ct}(x_i)$. The PALISADE library release 1.11.5 only supports ciphertext rotations for BGV and CKKS.

[7] The SEAL library release 3.7 does not offer the evaluation-at-index functionality, and only supports the noise value evaluation for BFV.

take 4.1ms for 8,192 integers, and a scalar-multiplication-based unmasking costs 9.5 ms or 9.8 ms for the BFV or BGV test, respectively.

Table 3 exhibits the multiplication saving owing to ModDiv in BFV and BGV. For k-1 times of ciphertext-wise multiplications (or multiplication then relinearization, denoted by $(k-1)\cdot(\mathsf{ct}\times\mathsf{ct})^*$ in Table 3), the straightforward BFV usage takes $(k-1)\cdot 131.5$ ms (or $(k-1)\cdot 132.1$ ms, respectively), which is reduced to 9.5 ms by ModDiv.

In addition, we compute the eight power of a ciphertext $(\mathsf{ct}(x))^8$ and the product of eight ciphertexts $\Pi_{i=1}^{8}\mathsf{ct}(x_i)$. These two tasks are common in any one or k variables, k degree polynomials in an FHE computation request. In BFV, invoking $\log_2(k)=3$ times of multiplication renders the 8 power of the ciphertext, which takes 398.1 ms in total. When $x_1,\cdots,x_k$ are stored in an 8-size batch, 7 times of evaluation at indexes and 7 multiplications cost 1219.4 ms in total. In BGV, we could take the rotation way to compute $\Pi_{i=1}^{8}\mathsf{ct}(x_i)$ if the messages are arranged in order in a batch, denoted by $\Pi_{i=1}^{8}\mathsf{ct}(x_i)^*$ in Table 3. The multiplication production tasks in BGV take several hundreds of milliseconds or seconds.

ModDiv prepares masked $\widetilde{x^k}$ or $\Pi_{i=1}^{k}\widetilde{x^i}$ outside an FHE scheme. The optimized computation (a single plaintext-ciphertext multiplication) overhead is 9.5 ms or 9.8 ms for computing both $(\mathsf{ct}(x))^8$ and $\Pi_{i=1}^{8}\mathsf{ct}(x_i)$ in BFV or BGV, respectively. The time reduction is considerable.

Similarly, Table 5 presents the introduced noise volume of the straightforward FHE usage and our protocol. Although a scalar multiplication requires 21 bits noise in our protocol, which is still much smaller than the one to compute $(\mathsf{ct}(x))^8$ (86 bits) and $\Pi_{i=1}^{8}\mathsf{ct}(x_i)$ (87 bits) in BFV. For a larger k value, the noise reduction by our protocol would also increase.

5.3 XOR-masking for CKKS and FHEW

We test our XOR-masking HEAD protocol for CKKS and FHEW, similarly selecting the cases for an 8,192-size batch and 8,192 unbatched ciphertexts.

Table 4 shows our storage optimization compared with the straightforward CKKS usage. The CKKS encryption parameters are configured by $n=16,384, \log_2 Q=150$, and we consider 16-bit fixed-point real numbers, and 8 bits represent the fraction part. When the batch size is 8,192, Trans (Step 1 in Fig. 6), i.e., generating $\{r_i\}$ and computing $x_i^j \oplus r_i^j$, costs 253 μs in total. The straightforward CKKS unbatched or batched ciphertext expansion rates are 525,872.5 and 64.2, respectively. While XOR-masking for CKKS still keeps a ciphertext expansion rate of 1. The computational overhead to unmask the 8,192 masked messages is relatively higher than the operations in other protocols, but is still reasonable, which consumes around 289.4 ms in total [8].

[8] The computations include sixteen times of encodings, scalar multiplications, and additions.

Table 5: Noise for ModSub and ModDiv HEAD (BFV parameters: $n=8,192$, $t=65,537$, $\log_2 Q=180$)

Scheme	Task	Straightforward FHE noise	ModSub or ModDiv noise
BFV	$\mathsf{ct}+\mathsf{pt}$	< 1 bit	< 1 bit
	$\mathsf{ct}+\mathsf{ct}$	< 1 bit	< 1 bit
	$\mathsf{pt}\cdot\mathsf{ct}$	21 bit	
	$\sum_{i=1}^{8}\mathsf{ct}(x)$	3 bits	
	$\sum_{i=1}^{8}\mathsf{ct}(x_i)^*$	5 bits	
	$(\mathsf{ct}(x))^8$	86 bits	21 bits
	$\Pi_{i=1}^{8}\mathsf{ct}(x_i)^*$	87 bits	

Table 6: Computation optimization for XOR-masking HEAD protocol (128 bits security for FHEW; XORs or XNORs: the computations for $\oplus_{i=1}^{k}\mathsf{ct}(x_i)$ or $\odot_{i=1}^{k}\mathsf{ct}(x_i)$, respectively)

Scheme	XORs	XNORs
Straightforward FHEW	$(k-1)\times 1.71$ s	$(k-1)\times 1.70$ s
XOR-masking HEAD	2 μs	2 μs

The implemented FHEW scheme in PALISADE only supports unpacked computations and symmetric encryption [9]. We configure the 128 bits security and leave other parameters as automatic recommendations. Table 4 shows that our protocol decreases the ciphertext expansion rate from $\frac{4141\text{ B}}{1\text{ bit}}=33,128$ to 1. The unmasking operation in the symmetric version of the XOR-masking protocol (step * in Fig. 7) reverses the value of $\mathsf{ct}(r_i)$ according to the value of the masking $\widetilde{x_i}$. Each ciphertext NOT operation takes 2 μs. If an 8,192-size masked vector $\{\widetilde{x_i}\}$ are all value 1, Table 4 shows the worst case that the unmasking operations take $8{,}192\times 2\mu$s. Our XOR-masking protocol also has a significant computation optimization for one or more ciphertext-wise XOR and XNOR operations. For the tasks of $\oplus_{i=1}^{k}\mathsf{ct}(x_i)$ and $\odot_{i=1}^{k}\mathsf{ct}(x_i)$, the overhead reduces from $(k-1)\cdot 1.7$s to 2μs.

6 Conclusion

In this paper, we presented HEAD, a generic FHE-based outsourced computation protocol for compact storage and efficient computation. We provide three HEAD instantiated protocols for different types of input messages. Our protocols accomplish a ciphertext expansion rate of 1 for storage and our experiments show the significant computation time and/or noise optimization for several evaluation functions.

[9] Another Boolean FHE library, tfhe [16] also only supports unpacked computations and symmetric encryption in the 1.0 version (https://tfhe.github.io/tfhe/releases.html).

References

[1] Fv-NFLlib. https://github.com/CryptoExperts/FV-NFLlib, May 2016. CryptoExperts.

[2] HEAAN. Online: https://github.com/snucrypto/HEAAN, September 2018. snucrypto.

[3] Microsoft SEAL (release 3.7). https://github.com/Microsoft/SEAL, September 2021. Microsoft Research, Redmond, WA.

[4] Lattigo v2.4.0. Online: https://github.com/ldsec/lattigo, January 2022. EPFL-LDS.

[5] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption security standard. Technical report, Toronto, Canada, November 2018.

[6] Pascal Aubry, Sergiu Carpov, and Renaud Sirdey. Faster homomorphic encryption is not enough: Improved heuristic for multiplicative depth minimization of boolean circuits. In *CT-RSA 2020*, pages 345–363.

[7] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles. In *TCC 2019*, pages 407–437.

[8] Zvika Brakerski, Craig Gentry, and Shai Halevi. Packed ciphertexts in lwe-based homomorphic encryption. In *PKC 2013*, pages 1–13.

[9] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory*, 6(3):13:1–13:36, 2014.

[10] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM J. Comput.*, 43(2):831–871, 2014.

[11] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptol.*, 13(1):143–202, 2000.

[12] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient homomorphic conversion between (ring) LWE ciphertexts. In *ACNS 2021*, pages 460–479.

[13] Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrède Lepoint, Mehdi Tibouchi, and Aaram Yun. Batch fully homomorphic encryption over the integers. In *EUROCRYPT 2013*, pages 315–335.

[14] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT 2017*, volume 10624, pages 409–437.

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

[16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption library, August 2016. https://tfhe.github.io/tfhe/.

[17] David Bruce Cousins, John Golusky, Kurt Rohloff, and Daniel Sumorok. An FPGA co-processor implementation of homomorphic encryption. In *IEEE HPEC 2014*, pages 1–6.

[18] David Bruce Cousins, Kurt Rohloff, and Daniel Sumorok. Designing an fpga-accelerated homomorphic encryption co-processor. *IEEE Trans. Emerg. Top. Comput.*, 5(2):193–206, 2017.

[19] Yarkin Doröz, Erdinç Öztürk, and Berk Sunar. Accelerating fully homomorphic encryption in hardware. *IEEE Trans. Computers*, 64(6):1509–1521, 2015.

[20] Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *EUROCRYPT 2015*, pages 617–640.

[21] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 144, 2012.

[22] Axel Feldmann, Nikola Samardzic, Aleksandar Krastev, Srini Devadas, Ronald G. Dreslinski, Karim Eldefrawy, Nicholas Genise, Chris Peikert, and Daniel Sánchez. F1: A fast and programmable accelerator for fully homomorphic encryption (extended version). *arXiv*, 2109.05371, 2021.

[23] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC 2009*, pages 169–178.

[24] Craig Gentry and Shai Halevi. Compressible FHE with applications to PIR. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019*, pages 438–464.

[25] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *EUROCRYPT 2011*, pages 129–148.

[26] Craig Gentry, Shai Halevi, Chris Peikert, and Nigel P. Smart. Field switching in bgv-style homomorphic encryption. *J. Comput. Secur.*, 21(5):663–684, 2013.

[27] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In *CRYPTO 2012*, volume 7417, pages 850–867.

[28] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the aes circuit (updated implementation). *IACR Cryptol. ePrint Arch.*, page 099, 2012.

[29] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO 2013*, pages 75–92.

[30] Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004.

[31] Shai Halevi and Victor Shoup. Algorithms in helib. In *CRYPTO 2014*, pages 554–571.

[32] Shai Halevi and Victor Shoup. Bootstrapping for helib. *J. Cryptol.*, 34(1):7, 2021.

[33] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha P. Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018*, pages 1651–1669.

[34] Kim Laine. Simple encrypted arithmetic library 2.3.1. https://www.microsoft.com/en-us/research/uploads/prod/2017/11/sealmanual-2-3-1.pdf, November 2017. Microsoft Research, Redmond, WA.

[35] KangHoon Lee and Ji Won Yoon. Efficient adaptation of TFHE for high end-to-end throughput. In *WISA 2021*, pages 144–156.

[36] Jie Li, Yamin Liu, and Shuang Wu. Pipa: Privacy-preserving password checkup via homomorphic encryption. In *ASIA CCS 2021*, pages 242–251.

[37] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT 1999*, pages 223–238. Springer.

[38] PALISADE (release 1.11.5). https://gitlab.com/palisade/palisade-release, September 2021.

[39] Brandon Reagen, Wooseok Choi, Yeongil Ko, Vincent T. Lee, Hsien-Hsin S. Lee, Gu-Yeon Wei, and David Brooks. Cheetah: Optimizing and accelerating homomorphic encryption for private inference. In *IEEE HPCA 2021*, pages 26–39.

[40] Sujoy Sinha Roy, Furkan Turan, Kimmo Järvinen, Frederik Vercauteren, and Ingrid Verbauwhede. Fpga-based high-performance parallel architecture for homomorphic computing on encrypted data. In *IEEE HPCA 2019*, pages 387–398.

[41] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Des. Codes Cryptogr.*, 71(1):57–81, 2014.

[42] Furkan Turan, Sujoy Sinha Roy, and Ingrid Verbauwhede. HEAWS: an accelerator for homomorphic encryption on the amazon AWS FPGA. *IEEE Trans. Computers*, 69(8):1185–1196, 2020.