Hybrid Framework for Approximate Computation over Encrypted Data

Jihoon Cho[2], Jincheol Ha[1], Seongkwang Kim[1], Joohee Lee[2], Jooyoung Lee[1], Dukjae Moon[2], and Hyojin Yoon[2]

[1] KAIST, Daejeon, Korea,
{smilecjf,ksg0923,hicalf}@kaist.ac.kr
[2] Samsung SDS, Seoul, Korea,
{jihoon1.cho,joohee1.lee,dukjae.moon,hj1230.yoon}@samsung.com

Abstract. Homomorphic encryption (HE) is a promising cryptographic primitive that enables computation over encrypted data, with various applications to medical, genomic, and financial tasks. In such applications, data typically contain some errors from their true values. The CKKS encryption scheme proposed by Cheon et al. (Asiacrypt 2017) supports approximate computation over encrypted data. However, HE schemes including CKKS commonly suffer from slow encryption speed and large ciphertext expansion compared to symmetric cryptography.

To address these problems, in particular, focusing on the client-side online computational overload and the ciphertext expansion, we propose a novel hybrid framework that supports CKKS. Since it seems to be infeasible to design a stream cipher operating on real numbers, we combine the CKKS and the FV homomorphic encryption schemes, and use a stream cipher using modular arithmetic in between. The proposed framework is thus dubbed the CKKS-FV transciphering framework. As a result, real numbers can be encrypted without significant ciphertext expansion or computational overload on the client side.

As a stream cipher to instantiate the CKKS-FV framework, we propose a new HE-friendly cipher, dubbed HERA, and analyze its security and efficiency. HERA is a stream cipher that features a simple randomized key schedule (RKS). Compared to recent HE-freindly ciphers such as FLIP and Rasta using randomized linear layers, HERA needs smaller number of random bits, leading to efficiency improvement on both the client and the server sides.

Our implementation shows that the CKKS-FV framework using HERA is 3.634 to 398 times faster on the client-side, compared to the environment where CKKS is only used, in terms of encryption time. Our framework also enjoys 2.4 to 436.7 times smaller ciphertext expansion according to the plaintext length.

Keywords: homomorphic encryption, transciphering framework, stream cipher, HE-friendly cipher

1 Introduction

Cryptography has been extensively used to protect data when it is stored (data-at-rest) or when it is being transmitted (data-in-transit). We also see increasing needs that data should be protected when it is being used, since it is often processed within untrusted environment. For example, organizations might want to migrate their computing environment from on-premise to public cloud, and to collaborate with their data without necessarily trusting each other. If data is protected by an encryption scheme which is *homomorphic*, then the cloud would be able to perform meaningful computations on the encrypted data, supporting a wide range of applications such as machine learning over a large amount of data preserving its privacy.

Homomorphic Encryption (for Approximate Computation). An encryption scheme that enables addition and multiplication over encrypted data without decryption key is called a *homomorphic encryption* (HE) scheme. Since the emergence of Gentry's blueprint [34], there has been a large amount of research in this area [12, 23, 24, 30, 35, 36, 38]. Various applications of HE to medical, genomic, and financial tasks have also been proposed [10, 20, 22, 43, 46, 51].

However, real-world data typically contain some errors from their true values since they are represented by real numbers rather than bits or integers. Even in the case that input data are represented by exact numbers without approximation, one might have to approximate intermediate values during data processing for efficiency. Therefore, it would be practically relevant to support approximate computation over encrypted data. To the best of our knowledge, the CKKS encryption scheme [21] is the only one that provides the desirable feature using an efficient encoder for real numbers. Due to this feature, CKKS achieves good performance in various applications, for example, to securely evaluate machine learning algorithms on a real dataset [11, 18, 52].

Unfortunately, HE schemes including CKKS commonly have two technical problems: slow encryption speed and large ciphertext expansion; the encryption/decryption time and the evaluation time of HE schemes are relatively slow compared to conventional encryption schemes. In particular, ciphertext expansion seems to be an intrinsic problem of homomorphic encryption due to the noise used in the encryption algorithm. Although the ciphertext expansion has been significantly reduced down to the order of hundreds in terms of the ratio of a ciphertext size to its plaintext size since the invention of the batching technique [35], it does not seem to be acceptable from a practical view point. Furthermore, this ratio becomes even worse when it comes to encryption of a short message; encryption of a single bit might result in a ciphertext of a few megabytes.

Transciphering Framework. To address the issue of the ciphertext expansion and the client-side computational overload, a hybrid framework, also called a *transciphering framework*, has been proposed [51] (see Figure 1). In the client-sever model, a client encrypts a message $\mathbf{m}$ using a symmetric cipher E with a

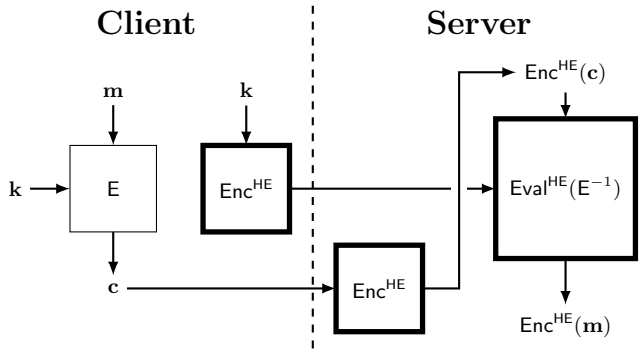


Fig. 1: The (basic) transciphering framework. Homomorphic operations are performed in the boxes with thick lines.

secret key $\mathbf{k}$; this secret key is also encrypted using an HE algorithm $\mathsf{Enc}^{\mathsf{HE}}$. The resulting ciphertexts $\mathbf{c} = \mathsf{E}_{\mathbf{k}}(\mathbf{m})$ and $\mathsf{Enc}^{\mathsf{HE}}(\mathbf{k})$ are stored in the server.

When the server wants to compute $\mathsf{Enc}^{\mathsf{HE}}(\mathbf{m})$ (for computation over encrypted data), it first computes $\mathsf{Enc}^{\mathsf{HE}}(\mathbf{c})$ for the corresponding ciphertext $\mathbf{c}$. Then the server homomorphically evaluates E^{-1} over $\mathsf{Enc}^{\mathsf{HE}}(\mathbf{c})$ and $\mathsf{Enc}^{\mathsf{HE}}(\mathbf{k})$, securely obtaining $\mathsf{Enc}^{\mathsf{HE}}(\mathbf{m})$.

Given a symmetric cipher with low multiplicative depth and complexity, this framework has the following advantages on the client side.

- A client does not need to encrypt all its data using an HE algorithm (except the symmetric key). All the data can be encrypted using only a symmetric cipher, significantly saving computational resources in terms of time and memory.
- Symmetric encryption does not result in ciphertext expansion, so the communication overload between the client and the server will be significantly low compared to using any homomorphic encryption scheme alone.

All these merits come at the cost of computational overload on the server side. That said, this trade-off would be worth considering in practice since servers are typically more powerful than clients.

A Symmetric Cipher over Real Numbers? The transciphering framework, as described above, does not directly apply to the CKKS scheme. The main reason is the difficulty in the design of an HE-friendly symmetric cipher E operating on real numbers. If a symmetric cipher E is given as a (complex) polynomial map, then any ciphertext will be represented by a polynomial in the corresponding plaintext and the secret key. Then, for given plaintext-ciphertext pairs $(\mathbf{m}_i, \mathbf{c}_i)$, an adversary will be able to establish a system of polynomial equations in the unknown key $\mathbf{k}$. The sum of $\|\mathsf{E}_{\mathbf{k}}(\mathbf{m}_i) - \mathbf{c}_i\|_2^2$ over the plaintext-ciphertext pairs also becomes a real polynomial, where the actual key is the zero of this function. Since this polynomial is differentiable, its (approximate) zeros will be efficiently

found by using iterative algorithms such as the gradient descent algorithm. By taking multiple plaintext-ciphertext pairs, the probability of finding any false key will be negligible.

HE-friendly Ciphers. Symmetric ciphers are built on top of linear and nonlinear layers, and in a conventional environment, there has been no need to take different design principles for the two types of layers with respect to their implementation cost. However, when a symmetric cipher is combined with BGV/FV-style HE schemes in a transciphering framework, homomorphic addition becomes way cheaper than homomorphic multiplication in terms of computation time and noise growth. With this observation, efficiency of an HE-friendly cipher is evaluated by its multiplicative complexity and depth. In an arithmetic circuit, its multiplicative complexity is represented by the number of multiplications (ANDs in the binary case). Multiplicative depth is the depth of the tree that represents the arithmetic circuit, closely related to the noise growth in the HE-ciphertexts. These two metrics have brought a new direction in the design of symmetric ciphers: to use simple nonlinear layers at the cost of highly randomized linear layers as adopted in the design of FLIP [50] and Rasta [27].

1.1 Our Contribution

The main constriction of this paper is two-fold. The first is to propose a new transciphering framework for the CKKS scheme that supports approximate computation over encrypted data. As discussed above, it seems to be infeasible to design a symmetric cipher over real numbers. In order to overcome this problem, we combine CKKS with FV which is a homomorphic encryption scheme using modular arithmetic [33], obtaining a novel hybrid framework, dubbed the CKKS-FV transciphering framework. This framework requires a symmetric cipher using modular arithmetic.

The second contribution is to propose a new stream cipher, dubbed HERA, to be built in our framework. The HERA cipher, operating on a modular space with a randomized key schedule, turns out to be faster than existing constructions in this line of research. With HERA combined with the CKKS-FV framework, real numbers can be encrypted without significant ciphertext expansion or computational overload on the client side.

Overview of the CKKS-FV Framework. Given a symmetric cipher E using modular arithmetic on $\mathbb{Z}_t$ ($t > 2$), the client encodes any message $\mathbf{m}$, which can be seen as a real number, into a vector in $\mathbb{Z}_t^N$, and then encrypts it using E. This "E-ciphertext" will be sent to the server and stored there. On the other hand, the secret key of E is encrypted by FV and also stored in the server.

Whenever a "CKKS-ciphertext" is needed for any message $\mathbf{m}$, the server encrypts the E-ciphertext of $\mathbf{m}$ again, using the FV scheme. With the resulting FV-ciphertext and the FV-encrypted key, the server homomorphically evaluates E^{-1}, obtaining the FV-ciphertext of encoded $\mathbf{m}$. Finally, this FV-ciphertext is translated into the corresponding CKKS-ciphertext of $\mathbf{m}$. Afterwards, the server

Message Length	Scheme	N	Ciphertext Expansion			Performance	
			Message	Ciphertext	Ratio	Latency	Throughput
Short	CKKS-FV	-	20 B	34 B	1.7	1.791 μs	10.65 MB/s
	CKKS (level 0)	2^{11}		14848 B	742.4	712.9 μs	27.40 KB/s
	CKKS (full level)	2^{11}		27648 B	1382.4	717.7 μs	27.21 KB/s
Long	CKKS-FV	-	40 KB	108 KB	2.7	4.673 ms	8.359 MB/s
	CKKS (level 0)	2^{15}		264 KB	6.6	16.98 ms	2.356 MB/s
	CKKS (full level)	2^{15}		6512 KB	162.8	158.2 ms	0.2528 MB/s

Table 1: Comparison of the CKKS-FV transciphering framework with HERA and the CKKS-only environment. All the experiments are done with 10-bit precision and 128-bit security.

will be able to approximately evaluate any circuit on the CKKS-ciphertexts. Details of this framework and the proof of its correctness are given in Section 3.

Why FV? In the FV scheme, a message is placed in the most significant bits of the ciphertext, while the error is in the least significant bits. So when an FV-ciphertext is decrypted by CKKS, the error still remains small without any blow-up.

The CKKS and FV schemes operate on a set of real numbers and a vector space over a finite field, say $\mathbb{Z}_t^N$, respectively. However, their encoding schemes map either type of messages to $\mathbb{Z}^N$. Furthermore, they use the same encryption algorithm. All these properties make FV an ideal candidate for an intermediate primitive between CKKS and a symmetric encryption algorithm.

Stream Ciphers Using Modular Arithmetic. In the CKKS-FV transciphering framework, a stream cipher using modular arithmetic is required. There are only a few ciphers using modular arithmetic [1, 4, 5, 37], and even such algorithms are not suitable for our transciphering framework due to their high multiplicative depths. In order to make our transciphering framework efficiently work, we propose a new HE-friendly cipher HERA, operating on a modular space with low multiplicative depth.

Recent constructions for HE-friendly ciphers such as FLIP and Rasta use randomized linear layers in order to reduce the multiplicative depth without security degradation. However, it seems that this type of ciphers require too many random bits in the generation of two-dimensional random matrices, slowing down the overall speed on both the client and the server sides. Instead of generating random matrices, we propose to randomize the key schedule algorithm by combining the secret key with a (public) random value for every round.

Implementations. The CKKS-FV framework is the first transciphering framework that supports approximate computation over encrypted data. So in this paper, our implementation is compared to the environment where the CKKS

scheme only is used, focusing on the ciphertext expansion and the client-side computational overload. The implementation results are summarized in Table 1.

In this table, the security parameter λ and the precision parameter p are set to 128 and 10, respectively. For CKKS, we measure the performance of two extreme sets of parameters, giving ciphertexts at level 0 and the full level, respectively. We note that our framework should be fairly compared to CKKS of level 0, since the CKKS-ciphertexts obtained at the end of the CKKS-FV framework should be bootstrapped for any subsequent computation over the ciphertexts. Even in this comparison, encryption of the CKKS-FV framework is 3.634 to 398 times faster than CKKS only (according to the message length). Our framework also suffers from ciphertext expansion due to the encoding phase, while it is still 2.4 to 436.7 times smaller than CKKS only (of level 0).

1.2 Related Work

The transciphering framework has first been proposed in [51]. In this framework, the circuit of the AES block cipher has been homomorphically evaluated [35]. This work was followed by the implementation of lightweight block ciphers SIMON [47] and PRINCE [29]. Since these ciphers have not been designed for the transciphering framework, the performance of the homomorphic evaluation was not satisfactory. In this line of research, low multiplicative complexity and depth becomes an important design principle, and LowMC is the first construction based on this design principle. However, it turned out that LowMC-80 and LowMC-128 are vulnerable to algebraic attacks and their variants [25, 28, 53].

Canteaut et al. claimed that stream ciphers might be advantageous in terms of online complexity compared to block ciphers, and proposed a new stream cipher Kreyvium [13]. However, its practical relevance is limited since the multiplicative depth (with respect to the secret key) keeps growing as keystreams are generated. A new stream cipher FLIP is based on a novel design strategy that its permutation layer is randomly generated for every encryption without increasing the algebraic degree in the secret key [50]. Rasta is a stream cipher aiming at higher throughput at the cost of high latency using random affine layers, which are determined by an extendable output function (XOF) [27].

Beside the transciphering framework, there are some attempt to reduce the memory overhead when encrypting short messages. Chen et al. proposed a conversion method between LWE ciphertexts and RLWE ciphertexts [16]. Small messages can be encrypted by LWE-based symmetric encryption with small ciphertext expansion, and a collection of LWE ciphertexts is converted to a RLWE ciphertext to perform a homomorphic evaluation. Chen et al. [17] proposed a hybrid SHE scheme using the CKKS packing algorithm and a variant of FV proposed by Bootland et al. [8]. This hybrid scheme makes the ciphertext size smaller compared to using CKKS only, in particular, when the number of slots is small.

2 Preliminaries

2.1 Notation

Throughout the paper, bold lowercase letters (resp. bold uppercase letters) denote vectors (resp. matrices). For a real number r, $\lfloor r \rceil$ denotes the nearest integer to r, rounding upwards in case of a tie. For an integer q, we identify $\mathbb{Z}_q$ with $\mathbb{Z} \cap (-q/2, q/2]$; for any integer z, $[z]_q$ denotes the mod q reduction of z into this interval. The notation $\lfloor \cdot \rceil$ and $[\cdot]_q$ are extended to vectors (resp. polynomials) to denote their component-wise (resp. coefficient-wise) reduction.

For a complex number z, its complex conjugate is denoted $\bar{z}$. This notation is also naturally extended to complex vectors. For a complex vector $\mathbf{x}$, its ℓ_p-norm is denoted $\|\mathbf{x}\|_p$. Throughout the paper, ζ and ξ denote a $2N$-th primitive root of unity over the complex field $\mathbb{C}$, and the finite field $\mathbb{Z}_t$, respectively, for fixed parameters N and t. The set of strings of arbitrary length over $\mathbb{Z}_t$ is denoted $\mathbb{Z}_t^*$. Usual dot products of vectors is denoted by $\langle \cdot, \cdot \rangle$. For two vectors (strings) $\mathbf{a}$ and $\mathbf{b}$, their concatenation is denoted $\mathbf{a} \| \mathbf{b}$.

For a set S, we will write $a \leftarrow S$ to denote that a is chosen from S uniformly at random. For a probability distribution $\mathcal{D}$, $a \leftarrow \mathcal{D}$ will denote that a is sampled according to the distribution $\mathcal{D}$. Unless stated otherwise, all logarithms are to the base 2.

2.2 Homomorphic Encryption

As the building blocks of our transciphering framework, we will briefly review the FV and CKKS homomorphic encryption schemes. For more details, we refer to [33, 21].

It is remarkable that FV and CKKS use the same ciphertext space; for a positive integer q, an integer M which is a power of two, and $N = M/2$, both schemes use

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(\Phi_M(X))$$

as their ciphertext spaces, where $\Phi_M(X) = X^N + 1$. They also use similar algorithms for key generation, encryption, decryption, and homomorphic addition and multiplication. However, the FV scheme supports *exact* computation modulo t (which satisfies $t \equiv 1 \pmod{M}$ throughout this paper), while the CKKS scheme supports *approximate* computations over the real numbers by taking different strategies to efficiently encode messages. We begin with their underlying hard problems.

LWE and RLWE. Let n and q be positive integers, and let $\mathcal{D}$ be a probability distribution over $\mathbb{Z}$. For an unknown vector $\mathbf{s} \in \mathbb{Z}_q^n$, the LWE (Learning with Errors) distribution $A_{n,q,\mathcal{D}}^{\mathsf{LWE}}(\mathbf{s})$ over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ is obtained by sampling a vector $\mathbf{a}$ uniformly at random from $\mathbb{Z}_q^n$ and an error e according to $\mathcal{D}$, and outputting

$$(\mathbf{a}, b = [\langle \mathbf{a}, \mathbf{s} \rangle + e]_q) \in \mathbb{Z}_q^n \times \mathbb{Z}_q.$$

The search-LWE problem is to find $\mathbf{s} \in \mathbb{Z}_q^n$ when independent samples $(\mathbf{a}_i, b_i)$ are obtained according to $A^{\mathsf{LWE}}_{n,q,\mathcal{D}}(\mathbf{s})$. The decision-LWE problem is to distinguish the distribution $A^{\mathsf{LWE}}_{n,q,\mathcal{D}}(\mathbf{s})$ from the uniform distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

Lyubashevsky et al. introduced the ring version of the LWE problem, which is also called Ring-LWE (RLWE) [48]. For a positive integer M, let $\Phi_M(X)$ be the M-th cyclotomic polynomial of degree $N = \phi(M)$. Let $\mathcal{R} = \mathbb{Z}[X]/(\Phi_M(X))$ and let $\mathcal{R}_q = \mathcal{R}/q\mathcal{R} = \mathbb{Z}_q[X]/(\Phi_M(X))$. The (decisional) RLWE problem is to distinguish the distribution of $(a, b = [a \cdot s + e]_q) \in \mathcal{R}_q^2$ from the uniform distribution over $\mathcal{R}_q^2$, where $s \in \mathcal{R}$ is a secret polynomial, a is sampled uniformly at random from $\mathcal{R}_q$ and e is sampled according to a certain error distribution over $\mathcal{R}$. The security of both FV and CKKS is based on the hardness assumption of the RLWE problem.

Encoders and Decoders. The main difference between FV and CKKS comes from their methods to encode messages lying in distinct spaces. The encoder $\mathsf{Ecd}^{\mathsf{FV}} : \mathbb{Z}_t^N \to \mathcal{R}$ of the FV scheme is the inverse of the decoder $\mathsf{Dcd}^{\mathsf{FV}}$ defined by, for $p(X) \in \mathcal{R}$,

$$\mathsf{Dcd}^{\mathsf{FV}}(p(X)) = [(p(\alpha_0), \cdots, p(\alpha_{N-1}))]_t \in \mathbb{Z}_t^N,$$

where $\alpha_i = \xi^{3^{i-1}} \pmod t$ for $0 \le i \le N-1$.[3]

Let δ be a positive real number (called a scaling factor in [21]). The CKKS encoder $\mathsf{Ecd}^{\mathsf{CKKS}} : \mathbb{C}^{N/2} \to \mathcal{R}$ is the (approximate) inverse of the decoder $\mathsf{Dcd}^{\mathsf{CKKS}} : \mathcal{R} \to \mathbb{C}^{N/2}$, where for $p(X) \in \mathcal{R}$,

$$\mathsf{Dcd}^{\mathsf{CKKS}}(p(X)) = \delta^{-1} \cdot (p(\beta_0), p(\beta_1), \cdots, p(\beta_{N/2-1})) \in \mathbb{C}^{N/2},$$

where $\beta_j = \zeta^{3^{j-1}} \in \mathbb{C}$ for $0 \le j \le N/2-1$.

Algorithms. FV and CKKS share a common key generation algorithm. The descriptions of those two algorithms have also been merged, so that one can easily compare the differences between FV and CKKS.

- Key generation: given a security parameter $\lambda > 0$, fix integers N, P, and q_0, $\cdots$, q_L such that q_i divides q_{i+1} for $0 \le i \le L-1$, and distributions $\mathcal{D}_{key}$, $\mathcal{D}_{err}$ and $\mathcal{D}_{enc}$ over $\mathcal{R}$ in a way that the resulting scheme is secure against any adversary with computational resource of $O(2^\lambda)$.
  1. Sample $a \leftarrow \mathcal{R}_{q_L}$, $s \leftarrow \mathcal{D}_{key}$, and $e \leftarrow \mathcal{D}_{err}$.
  2. The secret key is defined as $sk = (1, s) \in \mathcal{R}^2$, and the corresponding public key is defined as $pk = (b, a) \in \mathcal{R}_{q_L}^2$, where $b = [-a \cdot s + e]_{q_L}$.
  3. Sample $a' \leftarrow \mathcal{R}_{P \cdot q_L}$ and $e' \leftarrow \mathcal{D}_{err}$.
  4. The evaluation key is defined as $evk = (b', a') \in \mathcal{R}_{P \cdot q_L}^2$, where $b' = [-a' \cdot s + e' + Ps']_{P \cdot q_L}$ for $s' = [s^2]_{q_L}$.
- Encryption: given a public key pk and a plaintext $m \in \mathcal{R}$,
  1. Sample $r \leftarrow \mathcal{D}_{enc}$ and $e_0, e_1 \leftarrow \mathcal{D}_{err}$.

[3] A primitive root of unity ξ exists if the characteristic t of the message space is an odd prime such that $t \equiv 1 \pmod M$.

2. Compute $\mathsf{Enc}(pk, 0) = [r \cdot pk + (e_0, e_1)]_{q_L}$.
   - For FV, $\mathsf{Enc}^{\mathsf{FV}}(pk, m) = [\mathsf{Enc}(pk, 0) + (\Delta \cdot [m]_t, 0)]_{q_L}$, where $\Delta = \lfloor q_L/t \rceil$.
   - For CKKS, $\mathsf{Enc}^{\mathsf{CKKS}}(pk, m) = [\mathsf{Enc}(pk, 0) + (m, 0)]_{q_L}$.

– Decryption: given a secret key $sk \in \mathcal{R}^2$ and a ciphertext $ct \in \mathcal{R}_{q_\ell}^2$,

$$\mathsf{Dec}^{\mathsf{FV}}(sk, ct) = \left\lfloor \frac{t}{q_\ell} [\langle sk, ct \rangle]_{q_\ell} \right\rceil;$$
$$\mathsf{Dec}^{\mathsf{CKKS}}(sk, ct) = [\langle sk, ct \rangle]_{q_\ell}.$$

– Addition: given ciphertexts ct_1 and ct_2 in $\mathcal{R}_{q_\ell}^2$, their sum is defined as

$$ct_{add} = [ct_1 + ct_2]_{q_\ell}.$$

– Multiplication: given ciphertexts $ct_1 = (b_1, a_1)$ and $ct_2 = (b_2, a_2)$ in $\mathcal{R}_{q_\ell}^2$ and an evaluation key evk, their product is defined as

$$ct_{mult} = [(d_0, d_1) + \lfloor P^{-1} \cdot d_2 \cdot evk \rceil]_{q_\ell},$$

where (d_0, d_1, d_2) is defined by $[(b_1 b_2, a_1 b_2 + a_2 b_1, a_1 a_2)]_{q_\ell}$ when using CKKS and $\left[\left\lfloor \frac{t}{q}(b_1 b_2, a_1 b_2 + a_2 b_1, a_1 a_2) \right\rceil\right]_{q_\ell}$ when using FV.

– Rescaling (Modulus switching): given a ciphertext $ct \in \mathcal{R}_{q_\ell}^2$ and $\ell' < \ell$, its rescaled ciphertext is defined as

$$\mathsf{Rescale}_{\ell \to \ell'}(ct) = \left[\left\lfloor \frac{q_{\ell'}}{q_\ell} \cdot ct \right\rceil\right]_{q_{\ell'}}.$$

3 CKKS-FV Transciphering Framework

In this section, we describe how the CKKS-FV transciphering framework works, and prove its correctness.

3.1 Specification

With a fixed security parameter λ, all the other parameters for the FV and CKKS schemes will be set accordingly, including the degree of the polynomial modulus N, the ciphertext moduli $\{q_i\}_{i=0}^{L}$ (used for both FV and CKKS), and the FV plaintext modulus t. With these fixed parameters, we will describe how the framework works, distinguishing four parts; stream cipher generation, initialization, client-side computation, and server-side computation (See Figure 2). The client-side and server-side computations are explained in Algorithm 1 and Algorithm 2, respectively.

Generation of Stream Ciphers. For an integer n that divides N, the CKKS-FV framework will use a stream cipher E that takes as input a secret key $\mathbf{k} \in \mathbb{Z}_t^n$ and outputs a keystream $\mathbf{v} \in \mathbb{Z}_t^n$. We require that an additional input $\mathbf{u} \in \mathbb{Z}_t^*$ determines a distinct instance of E, denoted $\mathsf{E}_{\mathbf{u}}$ (or simply E).

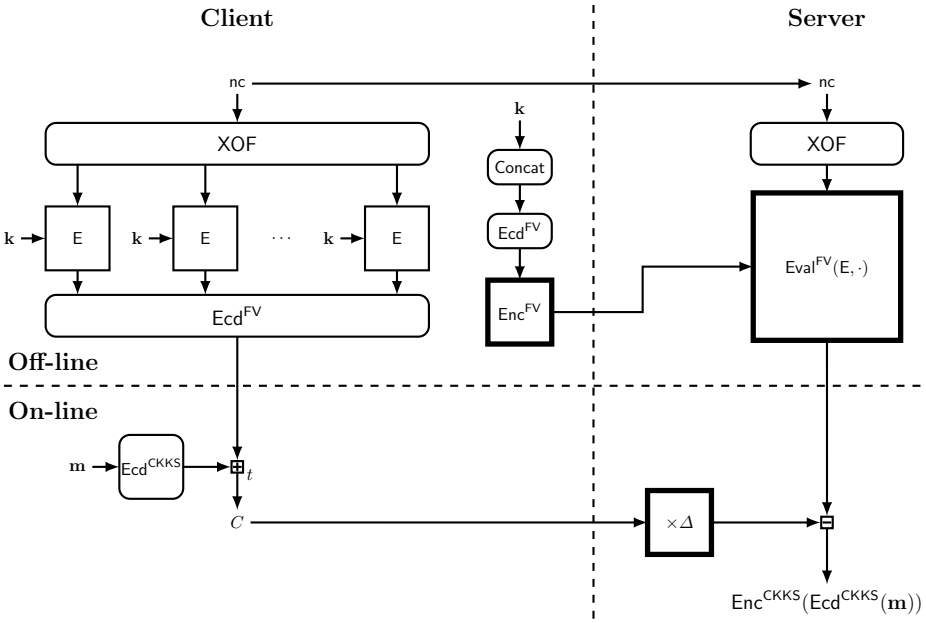


Fig. 2: The CKKS-FV transciphering framework. Homomorphic encryption and evaluation is performed in the boxes with thick lines. Operations in the boxes with rounded corners do not use any secret information. The vertical dashed line distinguishes the client-side and the server-side computation, while the horizontal dashed line distinguishes the offline and the online computation.

Generation of an instance of E by $\mathbf{u} \in \mathbb{Z}_t^*$ is denoted Gen (resp. Gen') in Algorithm 1 (resp. Algorithm 2). The input $\mathbf{u}$ is again generated by the underlying extendable output function (XOF),

$$\mathsf{XOF} : \{0,1\}^\lambda \times \mathbb{Z} \to \mathbb{Z}_t^*$$

that takes as inputs a public random value $\mathsf{nc} \in \{0,1\}^\lambda$ and a counter $\mathsf{ctr} \in \{1, \ldots, N/n\}$, and returns a string of elements of $\mathbb{Z}_t$. We will instantiate a pair of E and XOF with HERA as described in Section 4, in which case the output length of XOF is determined by the number and the size of the round keys of E.

Initialization. We use FV and CKKS with the same cyclotomic polynomial of degree N, and the same public-private key pair (pk, sk). The public key pk is shared by the server and the client.

The client encrypts $\mathbf{k} \in \mathbb{Z}_t^n$ using the FV scheme with pk. A packing technique might allow one to perform parallel computations for multiple messages encrypted in one ciphertext in a SIMD (Single Instruction, Multiple Data) manner. Hence, it is desirable to find an efficient packing method to homomorphically evaluate multiple copies of E on $\mathbf{k}$, depending on the choice of E.

For a matrix

$$\mathsf{Concat}(\mathbf{k}) \coloneqq \underbrace{(\mathbf{k}\|\mathbf{k}\|\cdots\|\mathbf{k})}_{k\text{-times}} \in \mathbb{Z}_t^{n\times k}$$

where the i-th column of $\mathsf{Concat}(\mathbf{k})$ is $\mathbf{k}$ and $k \leq N$, (glued) row-wise or column-wise packing methods can be used to encrypt it. We take the glued column-wise packing for $k = N/n$ and encrypt it to obtain a single ciphertext on the client side. For an efficient implementation, we use row-wise packing on the server side where $k = N$, which outputs n HE-ciphertexts concurrently. After homomorphic evaluation, the server re-aligns the n HE-ciphetexts into glued column-wise packed n ciphertexts to compute them with the output ciphertext of the client. Detailed description for row-wise and column-wise packing can be found in Appendix A. To summarize, the client computes

$$\mathcal{K} \coloneqq \mathsf{Enc}^{\mathsf{FV}}(pk, \mathsf{Ecd}^{\mathsf{FV}}(\mathsf{Concat}(\mathbf{k}))),$$

and sends $\mathcal{K}$ to the server. We note that this initialization phase can be done only once at the beginning of the CKKS-FV framework. The client also generates a random value $\mathsf{nc} \in \{0,1\}^\lambda$ and sends it to the server.

Client-side Computation. Given a nonce $\mathsf{nc} \in \{0,1\}^\lambda$, a secret key $\mathbf{k} \in \mathbb{Z}_t^n$ of E, an $N/2$-tuple of complex messages $\mathbf{m} = (m_1, \ldots, m_{N/2}) \in \mathbb{C}^{N/2}$, and a scaling factor $\delta > 0$ (used in the CKKS scheme), the client executes the following two steps.

Step 1: Keystream Generation (Offline). For each counter $\mathsf{ctr} \in \{1, \ldots, N/n\}$, the client computes $\mathbf{u}_{\mathsf{ctr}} \coloneqq \mathsf{XOF}(\mathsf{nc}, \mathsf{ctr})$, and generates the corresponding stream cipher E by procedure Gen in Algorithm 1; with this stream cipher E and secret key $\mathbf{k}$, the client computes $\mathbf{v}_{\mathsf{ctr}} \coloneqq \mathsf{E}(\mathbf{k})$. With $\mathbf{v}_1, \ldots, \mathbf{v}_{N/n} \in \mathbb{Z}_t^n$, the client computes a keystream

$$V \coloneqq \mathsf{Ecd}^{\mathsf{FV}}(\mathbf{v}_1, \ldots, \mathbf{v}_{N/n}) \in \mathbb{Z}_t^N.$$

Step 2: Message Encryption (Online). The client encodes the tuple of messages $\mathbf{m} = (m_1, \ldots, m_{N/2}) \in \mathbb{C}^{N/2}$ into $\mathcal{R}$ with the CKKS encoder equipped with the scaling factor δ. Using the correspondence between $\mathcal{R}$ and $\mathbb{Z}^N$, the client computes

$$C \coloneqq \left[\mathsf{Ecd}^{\mathsf{CKKS}}(\mathbf{m}) + V\right]_t,$$

and sends it to the server.

Server-side Computation. Given a nonce $\mathsf{nc} \in \{0,1\}^\lambda$, the FV-encrypted key $\mathcal{K} = \mathsf{Enc}^{\mathsf{FV}}(pk, \mathsf{Ecd}^{\mathsf{FV}}(\mathsf{Concat}(\mathbf{k})))$ and the symmetric ciphertext C, the server executes the following two steps.

Step 1: Homomorphic Evaluation (Offline). The server is able to recover $\mathbf{u}_{\mathsf{ctr}} = \mathsf{XOF}(\mathsf{nc}, \mathsf{ctr})$ (using the nonce nc sent from the client), and generate the stream cipher $\mathsf{E} \leftarrow \mathsf{Gen}(\mathbf{u}_{\mathsf{ctr}})$ for $\mathsf{ctr} = 1, \ldots, N/n$. Then, it constructs a circuit for the

homomorphic evaluation of N/n copies of E using the SIMD operation, denoted $\mathsf{Eval}^{\mathsf{FV}}(\mathsf{E}, \cdot)$. The procedure of generating the stream cipher and constructing a circuit for $\mathsf{Eval}^{\mathsf{FV}}(\mathsf{E}, \cdot)$ is denoted Gen' in Algorithm 2. With the FV-encrypted key $\mathcal{K}$, the server homomorphically computes $\mathcal{V} \coloneqq \mathsf{Eval}^{\mathsf{FV}}(\mathsf{E}, \mathcal{K})$.

Step 2: Retrieval of the CKKS*-ciphertext (Online).* The server computes a trivial FV-encryption of C to enable FV evaluation, namely

$$\mathcal{C} \coloneqq (\Delta \cdot C, 0).$$

Then, it computes

$$\mathcal{M} \coloneqq [\mathcal{C} - \mathcal{V}]_q,$$

where q is the ciphertext modulus of $\mathcal{V}$. In Section 3.2, we will show that the output $\mathcal{M}$ can be interpreted as a CKKS ciphertext of the client's message $\mathbf{m}$ indeed.

Algorithm 1: Client-side symmetric key encryption

Input:

- Nonce $\mathsf{nc} \in \{0,1\}^\lambda$
- Symmetric key $\mathbf{k} \in \mathbb{Z}_t^n$
- Tuple of messages $\mathbf{m} = (m_1, \ldots, m_{N/2}) \in \mathbb{C}^{N/2}$
- Scaling factor δ

Output:

- Symmetric ciphertext $C \in \mathcal{R}_t$

for $\mathsf{ctr} \leftarrow 1$ **to** N/n **do**
  $\mathbf{u}_{\mathsf{ctr}} \in \mathbb{Z}_t^* \leftarrow \mathsf{XOF}(\mathsf{nc}, \mathsf{ctr})$
  $\mathsf{E} \leftarrow \mathsf{Gen}(\mathbf{u}_{\mathsf{ctr}})$
  $\mathbf{v}_{\mathsf{ctr}} \leftarrow \mathsf{E}(\mathbf{k})$
$V \leftarrow \mathsf{Ecd}^{\mathsf{FV}}(\mathbf{v}_1, \ldots, \mathbf{v}_{N/n})$
$M \leftarrow \mathsf{Ecd}^{\mathsf{CKKS}}(\mathbf{m}, \mathrm{scale} = \delta)$
$C \leftarrow [M + V]_t$
return C

Security of the CKKS**-**FV **Framework.** In the server, all the client's data are encrypted using the stream cipher E, and the secret key is also encrypted by the FV encryption scheme.

In our framework, the underlying XOF will be modeled as a random oracle, and we will assume that E behaves like an independent random function for a random input string $\mathbf{u} \in \mathbb{Z}_t^*$ which is an output of XOF. Hence, the stream

cipher E will generate an independent random keystream by every distinct pair of a nonce and a counter. Keystreams from E are encoded by the encoder of FV, while it does not degrade the overall security since the encoder, being one-to-one, does not reduce the entropy of the keystreams.

Encryption of Short Messages. Since the parameter N is fixed according to the required depths for FV and CKKS in the initialization phase, it occurs that one needs to encrypt a shorter message than that in $\mathbb{C}^{N/2}$. In this case, a slight tweak to the above algorithms offers better performance in terms of the client-side computational overload and the ciphertext expansion.

Suppose that the message dimension is $\ell/2$ for a positive integer ℓ, where ℓ is a power-of-two such that $n \leq \ell < N$, namely, $\mathbf{m} \in \mathbb{C}^{\ell/2}$. Then one can first encode the message $\mathbf{m}$ of $\ell/2$ slots with the encoder $\mathsf{Ecd}_\ell^{\mathsf{CKKS}} : \mathbb{C}^{\ell/2} \to \mathbb{Z}[X]/(\Phi_{2\ell}(X))$ and then map it into the plaintext space $\mathcal{R}$ of the HE schemes using a function ψ defined by

$$\begin{aligned} \psi : \mathbb{Z}[X]/(\Phi_{2\ell}(X)) &\to \mathcal{R} = \mathbb{Z}[X]/(\Phi_{2N}(X)) \\ M(X) &\mapsto M(X^{N/\ell}), \end{aligned}$$

so that the resulting polynomial can be encrypted with FV.

Similarly, for any $\mathbf{v}_1, \ldots, \mathbf{v}_{\ell/n} \in \mathbb{Z}_t^n$, one can first obtain

$$\mathsf{Ecd}_\ell^{\mathsf{FV}}(\mathbf{v}_1, \ldots, \mathbf{v}_{\ell/n}),$$

and then apply ψ so that the resulting polynomial is in the plaintext space of FV. In this way, we obtain the ciphertext

$$C_\ell = \left[\mathsf{Ecd}_\ell^{\mathsf{CKKS}}(\mathbf{m}) + \mathsf{Ecd}_\ell^{\mathsf{FV}}(\mathbf{v}_1, \ldots, \mathbf{v}_{\ell/n})\right]_t,$$

instead of C defined in line 7 of Algorithm 1. Upon receiving C_ℓ, the server maps it into $\mathcal{R}$ and encrypts it with FV using the cyclotomic polynomial of degree N. The remaining procedures are the same as Algorithms 1 and 2.

This tweak preserves the functionality of our $\mathsf{CKKS\text{-}FV}$ framework, while reducing the ciphertext size to $\ell\lceil \log t \rceil$, which is ℓ/N times smaller than the main version. The computational cost in the client side will be reduced by the same order.

3.2 Correctness of the Framework

In this section, we prove the correctness of the $\mathsf{CKKS\text{-}FV}$ framework. Precisely, we will prove that the output $\mathcal{M}$ from Algorithm 2 can be interpreted as $\mathsf{Enc}^{\mathsf{CKKS}}(pk, \mathsf{Ecd}^{\mathsf{CKKS}}(\mathbf{m}))$, namely, $\mathbf{m}$ is close to $\mathsf{Dcd}^{\mathsf{CKKS}}(\mathsf{Dec}^{\mathsf{CKKS}}(sk, \mathcal{M}))$ up to a small error with high probability. In the following theorem, we omit the notations of pk and sk in the HE algorithms for simplicity.

Theorem 1. *Let $\mathbf{m} \in \mathbb{C}^{N/2}$ be the client's message as an input to Algorithm 1 such that $M = \mathsf{Ecd}^{\mathsf{CKKS}}(\mathbf{m}, \mathrm{scale} = \delta)$ satisfies $\|M\|_\infty \leq \lfloor t/2 \rfloor$, and let $\mathcal{M}$ be the*

Algorithm 2: Server-side homomorphic evaluation of decryption

Input:

- Nonce $\mathsf{nc} \in \{0,1\}^\lambda$
- FV-encrypted key $\mathcal{K} = \mathsf{Enc}^{\mathsf{FV}}\left(\mathsf{Ecd}^{\mathsf{FV}}\left(\mathsf{Concat}(\mathbf{k})\right)\right)$
- Symmetric ciphertext $C \in \mathcal{R}_t$

Output:

- CKKS-ciphertext $\mathcal{M} = \mathsf{Enc}^{\mathsf{CKKS}}(\mathsf{Ecd}^{\mathsf{CKKS}}(\mathbf{m}))$ with scaling factor $\delta\Delta$

for $\mathsf{ctr} \leftarrow 1$ **to** N/n **do**
  $\mathbf{u}_{\mathsf{ctr}} \in \mathbb{Z}_t^* \leftarrow \mathsf{XOF}(\mathsf{nc}, \mathsf{ctr})$
$\mathsf{Eval}^{\mathsf{FV}}(\mathsf{E}, \cdot) \leftarrow \mathsf{Gen}'(\mathbf{u}_1, \ldots, \mathbf{u}_{N/n})$
$\mathcal{V} \leftarrow \mathsf{Eval}^{\mathsf{FV}}(\mathsf{E}, \mathcal{K})$
$\mathcal{C} \leftarrow (\Delta \cdot C, 0)$
$\mathcal{M} \leftarrow [\mathcal{C} - \mathcal{V}]_q$
return $\mathcal{M}$

output from Algorithm 2. If the ciphertext after the homomorphic evaluation of $\mathsf{Eval}^{\mathsf{FV}}(\mathsf{E}, \cdot)$ *has a decryption error* $e_{\mathsf{eval}} \in \mathcal{R}$ *such that* $\|e_{\mathsf{eval}}\|_\infty^{can} < B_{\mathsf{Eval}}$ *where* $\|\cdot\|_\infty^{can} := \|\mathsf{Dcd}^{\mathsf{CKKS}}(\cdot, \mathrm{scale} = 1)\|_\infty$*, then we have*

$$\left\|\mathbf{m} - \mathsf{Dcd}^{\mathsf{CKKS}}\left(\mathsf{Dec}^{\mathsf{CKKS}}(\mathcal{M}), \mathrm{scale} = \Delta\delta\right)\right\|_\infty \leq \frac{N}{2\delta} + \frac{1}{\Delta\delta}\left(\frac{Nt}{2} + B_{\mathsf{Eval}}\right).$$

Proof. Recall that, in Algorithm 1, $M = \mathsf{Ecd}^{\mathsf{CKKS}}(\mathbf{m}, \mathrm{scale} = \delta)$, and $V = \mathsf{Ecd}^{\mathsf{FV}}(\mathbf{v}_1, \cdots, \mathbf{v}_{N/n})$, where $\mathsf{E} = \mathsf{Gen}(\mathbf{u}_{\mathsf{ctr}})$, $\mathbf{v}_{\mathsf{ctr}} = \mathsf{E}(\mathbf{k})$, and $C = [M + V]_t$. In Algorithm 2, we have

$$\mathcal{K} = \mathsf{Enc}^{\mathsf{FV}}\left(\mathsf{Ecd}^{\mathsf{FV}}(\mathbf{k}\|\cdots\|\mathbf{k})\right), \qquad \mathcal{V} = \mathsf{Eval}^{\mathsf{FV}}(\mathsf{E}, \mathcal{K}),$$
$$\mathcal{C} = (\Delta \cdot C, 0), \qquad \mathcal{M} = [\mathcal{C} - \mathcal{V}]_q.$$

Since

$$\mathsf{Dec}^{\mathsf{CKKS}}(\mathcal{V}) = \Delta V + e_{\mathsf{eval}},$$
$$\mathsf{Dec}^{\mathsf{CKKS}}(\mathcal{C}) = \Delta[M + V]_t,$$

we have

$$\begin{aligned}\mathsf{Dec}^{\mathsf{CKKS}}([\mathcal{C}-\mathcal{V}]_q) &= [\mathsf{Dec}^{\mathsf{CKKS}}(\mathcal{C}) - \mathsf{Dec}^{\mathsf{CKKS}}(\mathcal{V})]_q \\ &= [\Delta([M+V]_t - [V]_t) - e_{\mathsf{eval}}]_q \\ &= [\Delta[M]_t + (\Delta t - q)\varepsilon - e_{\mathsf{eval}}]_q\,, \end{aligned} \tag{1}$$

where $\varepsilon \in \mathcal{R}$ satisfies that $[M+V]_t - ([M]_t + [V]_t) = t\varepsilon$, and hence $\|\varepsilon\|_\infty \le 1$. Since $\|\mathsf{Dcd}^{\mathsf{CKKS}}(M, \mathrm{scale} = \delta) - \mathbf{m}\|_\infty \le \frac{N}{2\delta}$ and $[M]_t = M$ by (1), we have

$$\begin{aligned}\left\|\mathbf{m} - \mathsf{Dcd}^{\mathsf{CKKS}}\left(\mathsf{Dec}^{\mathsf{CKKS}}(\mathcal{M}, \mathrm{scale} = \Delta\delta)\right)\right\|_\infty &\le \frac{N}{2\delta} + \frac{|\Delta t - q| \cdot \|\varepsilon\|_\infty^{can} + \|e_{\mathsf{eval}}\|_\infty^{can}}{\Delta\delta} \\ &\le \frac{N}{2\delta} + \frac{|\Delta t - q| N + \|e_{\mathsf{eval}}\|_\infty^{can}}{\Delta\delta} \\ &\le \frac{N}{2\delta} + \frac{1}{\Delta\delta}\left(\frac{Nt}{2} + B_{\mathsf{Eval}}\right). \end{aligned}$$

□

4 A New Stream Cipher over $\mathbb{Z}_t$

The CKKS-FV transciphering framework requires a stream cipher with a variable plaintext modulus. In this section, we propose a new stream cipher HERA using modular arithmetic, and analyze its security.

4.1 Specification

A stream cipher HERA for λ-bit security takes as input a symmetric key $\mathbf{k} \in \mathbb{Z}_t^{16}$, a nonce $\mathsf{nc} \in \{0,1\}^\lambda$, and returns a keystream $\mathbf{k}_{\mathsf{nc}} \in \mathbb{Z}_t^{16}$, where the nonce is fed to the underlying extendable output function (XOF) that outputs an element in $(\mathbb{Z}_t^{16})^*$. In a nutshell, HERA is defined as follows.

$$\mathsf{HERA}[\mathbf{k}, \mathsf{nc}] = \mathsf{Fin}[\mathbf{k}, \mathsf{nc}, r] \circ \mathsf{RF}[\mathbf{k}, \mathsf{nc}, r-1] \circ \cdots \circ \mathsf{RF}[\mathbf{k}, \mathsf{nc}, 1] \circ \mathsf{ARK}[\mathbf{k}, \mathsf{nc}, 0]$$

where the i-th round function $\mathsf{RF}[\mathbf{k}, \mathsf{nc}, i]$ is defined as

$$\mathsf{RF}[\mathbf{k}, \mathsf{nc}, i] = \mathsf{ARK}[\mathbf{k}, \mathsf{nc}, i] \circ \mathsf{Cube} \circ \mathsf{MixRows} \circ \mathsf{MixColumns}$$

and the final round function Fin is defined as

$$\begin{aligned}&\mathsf{Fin}[\mathbf{k}, \mathsf{nc}, r] = \\ &\quad \mathsf{ARK}[\mathbf{k}, \mathsf{nc}, r] \circ \mathsf{MixRows} \circ \mathsf{MixColumns} \circ \mathsf{Cube} \circ \mathsf{MixRows} \circ \mathsf{MixColumns}\end{aligned}$$

for $i = 1, 2, \ldots, r-1$ (see Figure 3).

Key Schedule. The round key schedule can be simply seen as component-wise product between a random value and the master key $\mathbf{k}$, where the uniformly random value in $\mathbb{Z}_t^\times$ is obtained from a certain extendable output function XOF.

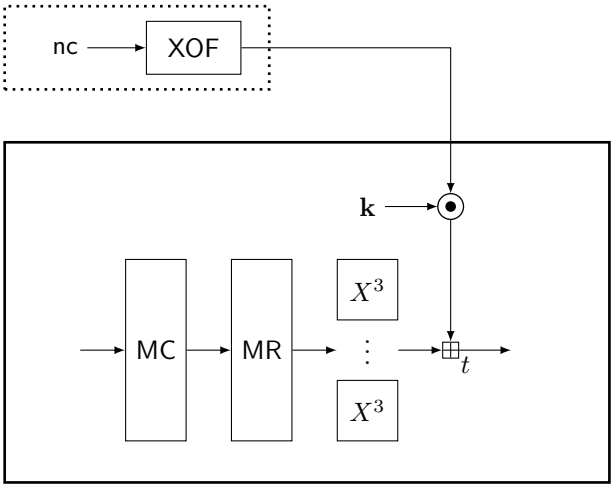


Fig. 3: The round function of HERA. Operations in the box with dotted (resp. thick) lines are public (resp. secret). "MC" and "MR" represent MixColumns and MixRows, respectively.

Given a sequence of outputs of XOF, say $\mathbf{rc} = (\mathbf{rc}_0, \ldots, \mathbf{rc}_r) \in (\mathbb{Z}_t^{16})^{r+1}$, ARK is defined as follows.

$$\mathsf{ARK}[\mathbf{k}, \mathsf{nc}, i](\mathbf{x}) = \mathbf{x} + \mathbf{k} \bullet \mathbf{rc}_i$$

for $i = 0, \ldots, r$, and $\mathbf{x} \in \mathbb{Z}_t^{16}$, where $\bullet$ (resp. $+$) denotes component-wise mod t multiplication (resp. addition). The extendable output function XOF might be instantiated with a sponge-type hash function SHAKE256 [32].

Linear Layers. Each linear layer is the composition of MixColumns and MixRows. Similarly to AES, MixColumns multiplies a certain 4×4-matrix to each column of the state, where the state of HERA is also viewed as a 4×4-matrix over $\mathbb{Z}_t$ (see Figure 4). MixColumns and MixRows are defined as in Figure 5a and Figure 5b, respectively. The only difference of our construction from AES is that each entry of the matrix is an element of $\mathbb{Z}_t$.

x_{00}	x_{01}	x_{02}	x_{03}
x_{10}	x_{11}	x_{12}	x_{13}
x_{20}	x_{21}	x_{22}	x_{23}
x_{30}	x_{31}	x_{32}	x_{33}

Fig. 4: State of HERA. Each square stands for the component in $\mathbb{Z}_t$.

$$\begin{bmatrix} y_{0c} \\ y_{1c} \\ y_{2c} \\ y_{3c} \end{bmatrix} = \begin{bmatrix} 2\,3\,1\,1 \\ 1\,2\,3\,1 \\ 1\,1\,2\,3 \\ 3\,1\,1\,2 \end{bmatrix} \cdot \begin{bmatrix} x_{0c} \\ x_{1c} \\ x_{2c} \\ x_{3c} \end{bmatrix}$$

(a) MixColumns

$$\begin{bmatrix} y_{c0} \\ y_{c1} \\ y_{c2} \\ y_{c3} \end{bmatrix} = \begin{bmatrix} 2\,3\,1\,1 \\ 1\,2\,3\,1 \\ 1\,1\,2\,3 \\ 3\,1\,1\,2 \end{bmatrix} \cdot \begin{bmatrix} x_{c0} \\ x_{c1} \\ x_{c2} \\ x_{c3} \end{bmatrix}$$

(b) MixRows

Fig. 5: Definition of MixColumns and MixRows. For $c \in \{0, 1, 2, 3\}$, x_{ij} and y_{ij} are defined as in Figure 4.

Nonlinear Layers. The nonlinear map Cube is the concatenation of 16 copies of the same S-box, where the S-box is defined by $x \mapsto x^3$ over $\mathbb{Z}_t$. So, for $\mathbf{x} = (x_0, \ldots, x_{15}) \in \mathbb{Z}_t^{16}$, we have

$$\mathsf{Cube}(\mathbf{x}) = (x_0^3, \ldots, x_{15}^3).$$

Encryption Mode. When a keystream of ℓ blocks (in $(\mathbb{Z}_t^{16})^\ell$) is needed for some $\ell > 0$, the "inner-counter mode" can be used; for $\mathsf{ctr} = 0, 1, \ldots, \ell - 1$, one computes

$$\mathbf{z}[\mathsf{ctr}] = \mathsf{HERA}\,[\mathbf{k}, \mathsf{nc} \| \mathsf{ctr}]\,(\mathbf{0}),$$

where $\mathbf{0}$ denotes the all-zero vector in $\mathbb{Z}_t^{16}$.

4.2 Design Rationale

Symmetric cipher designs for advanced protocols so far have been targeted at homomorphic encryption as well as various privacy preserving protocols such as multiparty computation (MPC) and zero knowledge proof (ZKP). In such protocols, multiplication is significantly more expensive than addition, so a new design principle has begun to attract attention in the literature: to use simple nonlinear layers at the cost of highly randomized linear layers (e.g., LowMC [3] and Rasta [27]). However, to the best of our knowledge, most symmetric ciphers following this new design principle operate only on binary spaces, rendering it difficult to apply them to our hybrid framework.

One might consider extending FLIP [50] or Rasta [27] to modular spaces by generating random matrices over modular spaces. This straightforward approach will degrade the overall efficiency of the cipher. Furthermore, unlike MPC and ZKP, linear maps over homomorphically encrypted data may not be simply "free". In order to use the batching techniques for homomorphic evaluation, the random linear layers should be encoded into HE-plaintexts, and then applied to HE-ciphertexts. Since multiplication between (encoded) plaintexts and ciphertexts require $O(N \log N)$ time (besides many HE rotations), randomized linear layers might not be that practical except that a small number of rounds are sufficient to mitigate algebraic attacks. For this reason, we opted for fixed linear layers.

In Table 2, we compare different types of linear maps to the (nonlinear) Cube map in terms of evaluation time and noise consumption. This experiment is conducted with the HE-parameters $(N, \lceil \log q \rceil) = (32768, 275)$ using row-wise packing, where the noise budget after the initialization is set to 239 bits (see Appendix A). In this table, "Fixed matrix" and "Freshly-generated matrix" represent a non-sparse fixed matrix, and a set of distinct matrices freshly generated over different slots, respectively, where all the matrices are square matrices of order 16 and randomly generated. We see that a freshly-generated linear layer takes more time than Cube. A fixed linear layer is better than a freshly-generated one, but its time complexity is not negligible yet compared to Cube. On the other hand, our linear layer is even faster than any fixed linear layer due to its sparsity.

	Time (ms)	Consumed Noise (bits)
MixRows $\circ$ MixColumns	23.55	4
Fixed matrix	461.68	27
Freshly-generated matrix	4006.03	34.9
Cube	3479.07	86.4

Table 2: Comparisons of different types of linear maps in terms of evaluation time and noise consumption.

The HERA cipher uses a sparse linear layer, whose design is motivated by the MixColumns layer in AES, enjoying a number of nice features; it is easy to analyze since its construction is based on an MDS matrix and needs a small number of multiplications due to the sparsity of the matrix. We design a $\mathbb{Z}_t$-variant of the matrix and use it in the linear layers; it turns out to be an MDS matrix over $\mathbb{Z}_t$ when t is a prime number such that $t > 17$. Instead of using ShiftRows of AES, HERA uses an additional layer MixRows which is a "row version" of MixColumns to enhance the security against algebraic attacks; the composition of two linear functions generates all possible monomials.

In the nonlinear layer, Cube takes the component-wise cube of the input. The cube map is studied from earlier multivariate cryptography [49], recently attracting renewed interest for the use in MPC-friendly ciphers [1, 4]. The cube map has good linear/differential characteristics, whose inverse is of high degree, mitigating meet-in-the-middle algebraic attacks.

As multiplicative depth heavily impacts on noise growth of HE-ciphertexts, it is desirable to design HE-friendly ciphers using a small number of rounds. One of the most threatening attacks on ciphers with low algebraic degrees is the higher order differential attack. For a λ-bit secure (possibly non-binary) cipher, the algebraic degree of the cipher should be at least $\lambda - 1$. However, the attack is not available on randomized cipher such as FLIP and Rasta.

To balance between the efficiency and the security, we propose a new direction: randomizing the key schedule. A randomized key schedule (RKS) is motivated by the tweakey framework [42]. In the tweakey framework, a key schedule takes as input a public value (called a tweak) and a key, where an adversary is allowed to take control of tweaks. On the other hand, RKS is a key schedule which takes as input a randomized public value and a key together, where the random value comes from a certain pseudorandom function. So, in our design, an adversary is not able to freely choose the random value.

The design principle behind our RKS is simple: to use as a small number of multiplications as possible. One might consider simply adding a fresh random value to the master key for every round. This type of key schedule might provide security against differential cryptanalysis, but it still might be vulnerable to algebraic attacks and linear cryptanalysis. It is important to enlarge the number of monomials in the first linear layer, while this candidate will invalidate this effect since an adversary is able to use the linear change of variables (see Appendix C in [27]). Based on this observation, we opted for component-wise multiplication. It simply offers better security on algebraic attacks and linear cryptanalysis.

4.3 Security Analysis

4.3.1 Algebraic Attacks

The HERA cipher can be represented by a set of polynomials over $\mathbb{Z}_t$ in unknowns $k_1, \ldots, k_n$, where $k_i \in \mathbb{Z}_t$ denotes the i-th component of the secret key $\mathbf{k} \in \mathbb{Z}_t^n$. Since multiplication is more expensive than addition in HE schemes, most HE-friendly ciphers have been designed to have a low multiplicative depth. This property might possibly make such ciphers vulnerable to algebraic attacks. Indeed, some of recent constructions have been analyzed by algebraic attacks due to their low algebraic depth [25, 31, 2]. In this section, we will consider two different types of algebraic attacks: trivial linearization and the Gröbner basis attack.

Trivial Linearization. Trivial linearization is to make the system of polynomial equations *linear* by replacing all monomials by new variables. When the cipher is represented by a system of polynomial equations of degree d over $\mathbb{Z}_t$ in n unknowns (and $d < t$), the number of monomials appearing in this system is upper bounded by

$$S = \sum_{i=0}^{d} \binom{n+i-1}{i}.$$

Therefore, at most S equations will be enough to solve this system of equations. If the system is sparse, then it would require less equations to solve the system. As shown in Supplementary Material A, all the cubic monomials appear in a single round of HERA so that all the monomials appear after r rounds of HERA. Therefore, this attack requires $O(S)$ data and $O(S^\omega)$ time, where $2 \leq \omega \leq 3$.

An adversary might take the *guess and determine* strategy before trivial linearization. By guessing g variables, the number of possible monomials is reduced

down to

$$S_g = \sum_{i=0}^{d} \binom{n-g+i-1}{i}.$$

This approach will be useful in particular when almost every monomial appears in the system. In this case, the overall time complexity becomes $O(t^g S_g^\omega)$.

Gröbner Basis Attack. The Gröbner basis attack is to solve a system of equations by computing a Gröbner basis of the system. If such a Gröbner basis is found, then the variables can be eliminated one by one. Gröbner basis can be computed with low data unlike the trivial linearization. However, its computation is slower than the trivial linearization with a small amount of data. For this reason, the Gröbner basis attack will be useful (compared to the trivial linearization) when either the data is limited or the number of monimials grows faster than the number of equations. When it comes to HERA, the Gröbner basis attack is mitigated by setting the parameters so that $O(S^\omega)$ is large enough.

Parameters. With respect to the algebraic attacks, the recommended number of rounds is given in Table 3 for a various security level. It has been computed by the above estimation for S^ω with $\omega = 2$. Guessing variables will not affect the security of HERA when $\lceil \log t \rceil \geq 17$.

Security (bit)	80	128	192	256
Round	4	5	6	7

Table 3: Recommended number of rounds with respect to algebraic attacks.

4.3.2 Linear and Differential Cryptanalysis

Linear Cryptanalysis. Linear cryptanalysis typically applies to block ciphers operating on binary spaces. However, linear cryptanalysis can be extended to non-binary spaces [6]; similarly to binary ciphers, for a prime t, the linear probability of a cipher $\mathsf{E} : \mathbb{Z}_t^\ell \to \mathbb{Z}_t^\ell$ with respect to input and output masks $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_t^\ell$ can defined as

$$\mathrm{LP}^{\mathsf{E}}(\mathbf{a}, \mathbf{b}) = \left| \mathbb{E}_{\mathbf{m}} \left[\exp\left\{ \frac{2\pi i}{t} \Big(- \langle \mathbf{a}, \mathbf{m} \rangle + \langle \mathbf{b}, \mathsf{E}(\mathbf{m}) \rangle \Big) \right\} \right] \right|^2,$$

where $\mathbf{m}$ follows the uniform distribution over $\mathbb{Z}_t^\ell$. When E is a random permutation, the expected linear probability is denoted

$$\mathrm{ELP}^{\mathsf{E}}(\mathbf{a}, \mathbf{b}) = \mathbb{E}_{\mathsf{E}}[\mathrm{LP}^{\mathsf{E}}(\mathbf{a}, \mathbf{b})].$$

Then the number of samples required for linear cryptanalysis is known to be

$$\frac{1}{\mathrm{ELP}^{\mathsf{E}}(\mathbf{a},\mathbf{b})}.$$

In order to ensure the security against linear cryptanalysis, it is sufficient to bound the maximum linear probability $\max_{\mathbf{a}\neq 0,\mathbf{b}} \mathrm{ELP}^{\mathsf{E}}(\mathbf{a},\mathbf{b})$.

The linear probability of an r-round HERA is upper bounded by $(\mathrm{LP}^S)^{B_\ell \cdot \lfloor \frac{r}{2} \rfloor}$, where LP^S and B_ℓ denote the linear probability of the S-box and the branch number of the linear layer, respectively. Therefore, the data complexity for linear cryptanalysis is lower bounded approximately by

$$\frac{1}{(\mathrm{LP}^S)^{B_\ell \cdot \lfloor \frac{r}{2} \rfloor}}.$$

The linear probability LP^S is upper bounded as follows.

Lemma 1. *For an odd prime t, let $S : \mathbb{Z}_t \to \mathbb{Z}_t$ be a permutation such that $S(x) = x^3$. Then, for any pair $(\alpha, \beta) \in \mathbb{Z}_t^2$ such that $\alpha \neq 0$, we have*

$$\mathrm{LP}^S(\alpha,\beta) \leq \frac{4}{t}.$$

Proof. By the definition of LP, we have

$$\mathrm{LP}^S(\alpha,\beta) = \left|\mathbb{E}_m\Big[-\alpha m + \beta S(m)\Big]\right|^2 = \left|\frac{1}{t}\sum_{m=0}^{t-1} \exp\left\{\frac{2\pi i}{t}\left(-\alpha m + \beta m^3\right)\right\}\right|^2.$$

Carlitz and Uchiyama [14] proved that

$$\left|\sum_{x=0}^{t-1} \exp\left(\frac{2\pi i}{t}\cdot p(x)\right)\right| \leq (r-1)\sqrt{t}$$

for any polynomial $p(x)$ of degree r over $\mathbb{Z}_t$. Therefore, we have

$$\mathrm{LP}^S(\alpha,\beta) = \left|\frac{1}{t}\sum_{m=0}^{t-1} \exp\left\{\frac{2\pi i}{t}\Big(-\alpha m + \beta m^3\Big)\right\}\right|^2 \leq \frac{1}{t^2}\left(2\sqrt{t}\right)^2 = \frac{4}{t}. \qquad \square$$

The branch number of the linear layer of HERA is 8 (as shown in Supplementary Material B). Combined with Lemma 1, we can conclude that an r-round HERA cipher provides λ-bit security against linear cryptanalysis when

$$\left(\frac{t}{4}\right)^{8\cdot\lfloor\frac{r}{2}\rfloor} > 2^\lambda.$$

Differential Cryptanalysis. Resistance of a substitution-permutation cipher to differential cryptanalysis is typically estimated by the maximum probability

of differential trails [7]. Let $S : \mathbb{Z}_t \to \mathbb{Z}_t$ be the nonlinear map (S-box) used in Cube. Given nonzero input and output differences α and β, the differential probability of S is defined by

$$\mathrm{DP}^S(\alpha, \beta) = \frac{1}{t} \cdot |\{x \in \mathbb{Z}_t | S(x + \alpha) - S(x) = \beta\}| .$$

So $\mathrm{DP}^S(\alpha, \beta)$ is determined by the number of solutions to $S(x + \alpha) - S(x) = \beta$, which is a quadratic equation in x since $S(x) = x^3$. Therefore, there are at most two solutions to this equation, which implies $\mathrm{DP}^S(\alpha, \beta) \leq \frac{2}{t}$.

Since the branch number of the linear layer of HERA is 8 (as shown in Supplementary Material B), an r-round HERA cipher provides λ-bit security against differential cryptanalysis when

$$\left(\frac{t}{2}\right)^{8 \cdot \lfloor \frac{r}{2} \rfloor} > 2^{\lambda}.$$

Parameters. Based on the evaluation given as above, the recommended number of rounds is summarized in Table 4 for a various security level. This table assumes that $\lceil \log t \rceil \geq 17$.

Security (bit)	80	128	192	256
Round	2	4	4	6

Table 4: Recommended number of rounds with respect to linear and differential cryptanalysis.

Remark 1. Our evaluation of the security of HERA against linear and differential cryptanalysis is based on the assumption that round keys are fixed under the same master key, while it is not the case for HERA. Due to its randomized key scheduling, HERA will have additional security margins against such types of attacks. We will choose the number of rounds with respect to the security against algebraic attacks, in which case we have a few more rounds of security margin for linear and differential cryptanalysis.

4.3.3 Related-key Attacks and Others.

Related-key Attacks. Kelsey et al. [44] have first proposed a related key attack on DES with independent round keys. They used 15 related keys with a partial collision property that all the round keys are the same except for a particular target round. However, it would be infeasible to find such a partial collision for HERA due to its randomized key schedule. Hence, the related key attack with

independent round keys will not be applicable to HERA. Related-key attacks with random relationship between rounds keys would not work better than the classical linear and differential cryptanalysis.

Other Attacks. Given a fixed polynomial representation of ciphertexts in plaintext variables, and if the polynomial is of a low degree, then one can mount the cube attack [26]. By interpolating a polynomial from plaintext-ciphertext pairs, one might also mount the interpolation attack [41]. Such attacks would not be applicable to HERA since round keys are fresh for every encryption.

The integral attack [45] exploits the integral property of the underlying permutation. In particular, small bijective S-boxes and insufficiently diffusive linear layers make the block cipher vulnerable to the integral attack. However, this attack is possible only when a sufficient number of queries are made for the same round keys since otherwise one cannot make the integral property propagate over the rounds. Overall, we conclude that HERA is secure against the integral attack.

5 Implementation

In this section, we evaluate the performance of the CKKS-FV framework combined with the HERA cipher in terms of encryption speed and ciphertext expansion. Our source codes are developed in C++17 with Microsoft `SEAL` ver. 3.4.5 [54] which includes FV and CKKS implementations. Our experiments are done in AMD Ryzen 7 2700X @ 3.70 GHz single-threaded with 64 GB memory, using GNU C++ 7.5.0 compliler in O3 optimization level. XOF is instantiated with SHAKE256 in `XKCP` [55]. In the client-side implementation, we use the AVX2 instruction set.

Selected Parameters. Sets of parameters used in our implementation are given in Table 5, where

- λ is the security parameter of the CKKS-FV framework;
- p is the bits of precision of the CKKS-FV framework;
- t is the plaintext modulus of HERA;
- r is the number of rounds of HERA;
- N is the degree of the polynomial modulus in the HE schemes;
- q is the ciphertext modulus of the HE schemes before evaluating E.

Since the `SEAL` library supports only the security level of 128 bits or more, we experiment Par-I and Par-II, which target 80-bit security, using the HE schemes with 128-bit security parameters. In Table 5, we assume that plaintexts are fully batched.

Given N and p, we will fix positive integers c_l and c_u such that $c_u/c_l \leq 4$ and for any message $\mathbf{m} = (m_1, \ldots, m_{N/2})$,

$$\min_{\substack{\mathrm{Re}(m_i)\neq 0 \\ \mathrm{Im}(m_i)\neq 0}} \{|\mathrm{Re}(m_i)|, |\mathrm{Im}(m_i)|\} \geq c_l,$$

	λ	p	SKE		HE	
			$\lceil \log t \rceil$	r	$\log N$	$\lceil \log q \rceil$
Par-I	80	10	27	4	15	495
Par-II	80	14	31	4	15	550
Par-III	128	10	27	5	15	605
Par-IV	128	14	31	5	15	660

Table 5: Selected sets of parameters used in our implementations.

and $\|\mathbf{m}\|_\infty \leq c_u$. Once c_l and c_u are fixed, we choose a scaling factor δ satisfying

$$\alpha \cdot \frac{N}{\delta c_l} < 2^{-p}$$

where $\alpha < 0.476$ (See Supplementary Material C), and then the plaintext modulus t satisfying

$$\delta c_u < \frac{t}{2}.$$

In the case of encrypting short messages, we can choose an alternative set of parameters as shown in Table 6. Here we only change the number of slots and the corresponding plain modulus t from Par-III (without any change to the other parameters). The parameter ℓ implies that the message space is $\mathbb{C}^{\ell/2}$.

	$\lceil \log t \rceil$	$\log \ell$
Par-IIIA	17	4
Par-IIIB	20	6
Par-IIIC	21	8
Par-IIID	23	10

Table 6: Sets of parameters for short message encryption.

5.1 Benchmarks

As shown in Table 7, we measure the performance of the CKKS-FV framework, distinguishing two different parts: the client-side and the server-side as separated in Figure 2. On the client-side, the latency includes time for generating pseudo-random numbers (needed to generate a single keystream in $\mathbb{Z}_t^N$), computation of E, FV-encoding, CKKS-encoding and vector addition over $\mathbb{Z}_t$. When we measure the throughput, we set the message length to Np bits. The extendable output

	Client-side		Server-side	
	Latency (ms)	Throughput (MB/s)	Latency (s)	Throughput (KB/s)
Par-I	4.507	8.666	32.63	9.752
Par-II	4.872	11.23	37.32	11.77
Par-III	4.673	8.359	51.69	6.633
Par-IV	5.056	10.82	57.97	8.194

Table 7: Performance of the CKKS-FV transciphering framework with HERA.

	$\log \ell$	Client-side		Ciphertext Expansion			
		Latency (μs)	Throughput (MB/s)	Message (B)	Symmetric (B)	CKKS (B)	Ratio
Par-IIIA	4	1.791	10.65	20	34	14848	436.7
Par-IIIB	6	7.912	9.642	80	160	14848	92.80
Par-IIIC	8	32.29	9.452	320	640	14848	23.20
Par-IIID	10	129.2	9.448	1280	2816	14848	5.272
Par-III	15	4695	8.320	40960	110592	270336	2.444

Table 8: Client-side performance and ciphertext expansion in the CKKS-FV framework when short messages are encrypted.

function is instantiated with SHAKE256 in `XKCP`. For the uniform sampling on $\mathbb{Z}_t$, we refer to [9].

The server-side part is implemented by using the `SEAL` library. The latency includes time for randomized key schedule, homomorphic evaluation of E, multiplication of the client's output by Δ, and the homomorphic subtraction. As mentioned in Section 3.1, we use column-wise packing on the client side and row-wise packing on the server side. When the server re-aligns from row-wise packing to column-wise packing, it outputs 16 (column-wise packed) HE-ciphertexts. We measure the latency until the first HE-ciphertext comes out, and measure the throughput until all the 16 HE-ciphertext come out. We note that our evaluation does not take into account key encryption since the encrypted key will be used over multiple sessions once it is computed. For the same reason, the initialization process of the HE schemes is not considered.

Table 8 shows the client-side performance and the ciphertext expansion in the case of encrypting short messages. When the number of slots is reduced, both encoders become faster super-linearly. On the other hand, for the server-side computation, latency will be the same as the fully batched one, and the throughput will be inversely proportional to the number of slots.

The ciphertext expansion can be evaluated by the underlying parameters and the message length, independent of the experiments. The bits of precision is counted in the message length; if a complex vector $\mathbf{m}$ of length $\ell/2$ has p bits of precision, then we will regard $\mathbf{m}$ as of size ℓp bits. For a vector over $\mathbb{Z}_t$ of length ℓ, its size is regarded as $\ell\lceil \log t \rceil$ bits. When it comes to the CKKS scheme, the parameter N should be at least 2048 bits no matter how short messages are encrypted; for short messages, the CKKS-ciphertext length cannot be proportional to the message length.

5.2 Discussion

At the end of the CKKS-FV framework, the server obtains CKKS-ciphertexts with scaling factor $\delta \cdot \Delta$, as shown in Theorem 1. When the server evaluates any function for the ciphertexts, rescaling will be necessary in order to balance the ciphertext modulus and the scaling factor. However, the level of the ciphertext will still remain too low to do any further operation over the ciphertexts.

When FV or BGV is used in the transciphering framework [3, 13, 27], one can take sufficiently large parameters for the HE scheme in order to obtain ciphertexts at a level high enough to do additional computations without bootstrapping. When it comes to the CKKS-FV framework, one should choose a large parameter t for the underlying symmetric cipher in addition to large HE parameters. For example, in order to obtain CKKS-ciphertexts of level 2 with 10 bits of precision in each slot on average, the parameter $\log t$ should be approximately set to 120.

Alternatively, bootstrapping allows a noisy ciphertext which is an output of the CKKS-FV framework to be refreshed to a cleaner state. Bootstrapping methods for the CKKS scheme have been actively studied [19, 15, 39], while all the methods use the sine function to approximate modulo q operation in the decryption circuit of the CKKS scheme as far as we know. To make accurate approximation, it should be the case that $\|\mathbf{m}\|_\infty \ll q$, where $\mathbf{m}$ is a plaintext capsuled in the ciphertext to be bootstrapped and q is the ciphertext modulus. Since the resulting CKKS-ciphertext has scaling factor $\delta\Delta = \delta \cdot \lfloor q/t \rceil$ in our case, t should be at least $(C \cdot \epsilon^{-1} \cdot q)^{\frac{1}{3}}$ to make further bootstrapping sound, where $C = \frac{2\pi^2}{3}$ and the approximation error between the modulo q operation and the sine function is upper bounded by $\epsilon > 0$.

References

[1] Albrecht, M., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology – ASIACRYPT 2016. vol. 10031, pp. 191–219. Springer (2016)

[2] Albrecht, M.R., Cid, C., Grassi, L., Khovratovich, D., Lüftenegger, R., Rechberger, C., Schofnegger, M.: Algebraic Cryptanalysis of STARK-Friendly Designs: Application to MARVELlous and MiMC. In: Galbraith, S.D., Moriai, S. (eds.)

Advances in Cryptology – ASIACRYPT 2019. vol. 11923, pp. 371–397. Springer (2019)
[3] Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015. vol. 9056, pp. 430–454. Springer, Berlin, Heidelberg (2015)
[4] Aly, A., Ashur, T., Ben-Sasson, E., Dhooghe, S., Szepieniec, A.: Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols. IACR Cryptology ePrint Archive, Report 2019/426 (2019), https://eprint.iacr.org/2019/426
[5] Ashur, T., Dhooghe, S.: MARVELlous: a STARK-Friendly Family of Cryptographic Primitives. IACR Cryptology ePrint Archive, Report 2018/1098 (2018), https://eprint.iacr.org/2018/1098
[6] Baignères, T., Stern, J., Vaudenay, S.: Linear Cryptanalysis of Non Binary Ciphers. In: Adams, C., Miri, A., Wiener, M. (eds.) Selected Areas in Cryptography. vol. 4876, pp. 184–211. Springer (2007)
[7] Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. In: Menezes, A.J., Vanstone, S.A. (eds.) Advances in Cryptology – CRYPTO '90. vol. 537, pp. 2–21. Springer (1991)
[8] Bootland, C., Castryck, W., Iliashenko, I., Vercauteren, F.: Efficiently Processing Complex-Valued Data in Homomorphic Encryption. Journal of Mathematical Cryptology **14**(1), 55 – 65 (2020)
[9] Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehle, D.: CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM. In: 2018 IEEE European Symposium on Security and Privacy (Euro S&P). pp. 353–367 (2018)
[10] Bos, J.W., Lauter, K., Naehrig, M.: Private predictive analysis on encrypted medical data. Journal of biomedical informatics **50**, 234–243 (2014)
[11] Boura, C., Gama, N., Georgieva, M., Jetchev, D.: Simulating Homomorphic Evaluation of Deep Learning Predictions. In: Dolev, S., Hendler, D., Lodha, S., Yung, M. (eds.) Cyber Security Cryptography and Machine Learning. vol. 11527, pp. 212–230. Springer (2019)
[12] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) Fully Homomorphic Encryption without Bootstrapping. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. p. 309–325. ACM (2012)
[13] Canteaut, A., Carpov, S., Fontaine, C., Lepoint, T., Naya-Plasencia, M., Paillier, P., Sirdey, R.: Stream ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression. Journal of Cryptology **31**(3), 885–916 (2018)
[14] Carlitz, L., Uchiyama, S.: Bounds for exponential sums. Duke mathematical Journal **24**(1), 37–41 (1957)
[15] Chen, H., Chillotti, I., Song, Y.: Improved Bootstrapping for Approximate Homomorphic Encryption. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2019. vol. 11477, pp. 34–54. Springer (2019)
[16] Chen, H., Dai, W., Kim, M., Song, Y.: Efficient Homomorphic Conversion Between (Ring) LWE Ciphertexts. IACR Cryptology ePrint Archive, Report 2020/015 (2020), https://eprint.iacr.org/2020/015
[17] Chen, H., Iliashenko, I., Laine, K.: When HEAAN Meets FV: a New Somewhat Homomorphic Encryption with Reduced Memory Overhead. IACR Cryptology ePrint Archive, Report 2020/121 (2020), https://eprint.iacr.org/2020/121
[18] Cheon, J.H., Kim, D., Kim, Y., Song, Y.: Ensemble Method for Privacy-Preserving Logistic Regression Based on Homomorphic Encryption. IEEE Access **6**, 46938–46948 (2018)

[19] Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for Approximate Homomorphic Encryption. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018. vol. 10820, pp. 360–384. Springer (2018)

[20] Cheon, J.H., Jeong, J., Lee, J., Lee, K.: Privacy-Preserving Computations of Predictive Medical Models with Minimax Approximation and Non-Adjacent Form. In: Brenner, M., Rohloff, K., Bonneau, J., Miller, A., Ryan, P.Y., Teague, V., Bracciali, A., Sala, M., Pintore, F., Jakobsson, M. (eds.) Financial Cryptography and Data Security. vol. 10323, pp. 53–74. Springer (2017)

[21] Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic Encryption for Arithmetic of Approximate Numbers. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017. vol. 10624, pp. 409–437. Springer (2017)

[22] Cheon, J.H., Kim, M., Lauter, K.: Homomorphic Computation of Edit Distance. In: Brenner, M., Christin, N., Johnson, B., Rohloff, K. (eds.) Financial Cryptography and Data Security. vol. 8976, pp. 194–212. Springer (2015)

[23] Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology – ASIACRYPT 2016. vol. 10031, pp. 3–33. Springer (2016)

[24] Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast Fully Homomorphic Encryption Over the Torus. Journal of Cryptology **33**(1), 34–91 (2020)

[25] Dinur, I., Liu, Y., Meier, W., Wang, Q.: Optimized Interpolation Attacks on LowMC. In: Iwata, T., Cheon, J.H. (eds.) Advances in Cryptology – ASIACRYPT 2015. vol. 9453, pp. 535–560. Springer (2015)

[26] Dinur, I., Shamir, A.: Cube Attacks on Tweakable Black Box Polynomials. In: Joux, A. (ed.) Advances in Cryptology – EUROCRYPT 2009. vol. 5479, pp. 278–299. Springer (2009)

[27] Dobraunig, C., Eichlseder, M., Grassi, L., Lallemand, V., Leander, G., List, E., Mendel, F., Rechberger, C.: Rasta: A Cipher with Low ANDdepth and Few ANDs per Bit. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018. vol. 10991, pp. 662–692. Springer (2018)

[28] Dobraunig, C., Eichlseder, M., Mendel, F.: Higher-Order Cryptanalysis of LowMC. In: Kwon, S., Yun, A. (eds.) Information Security and Cryptology – ICISC 2015. vol. 9558, pp. 87–101. Springer (2016)

[29] Doröz, Y., Shahverdi, A., Eisenbarth, T., Sunar, B.: Toward Practical Homomorphic Evaluation of Block Ciphers Using Prince. In: Böhme, R., Brenner, M., Moore, T., Smith, M. (eds.) Financial Cryptography and Data Security. vol. 8438, pp. 208–220. Springer (2014)

[30] Ducas, L., Micciancio, D.: FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015. vol. 9056, pp. 617–640. Springer (2015)

[31] Duval, S., Lallemand, V., Rotella, Y.: Cryptanalysis of the FLIP Family of Stream Ciphers. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology – CRYPTO 2016. vol. 9814, pp. 457–475. Springer (2016)

[32] Dworkin, M.J.: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Tech. rep., National Institute of Standards and Technology (2015)

[33] Fan, J., Vercauteren, F.: Somewhat Practical Fully Homomorphic Encryption. IACR Cryptology ePrint Archive, Report 2012/144 (2012), https://eprint.iacr.org/2012/144

[34] Gentry, C.: Fully Homomorphic Encryption Using Ideal Lattices. In: Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing. p. 169–178. ACM (2009)
[35] Gentry, C., Halevi, S., Smart, N.P.: Homomorphic Evaluation of the AES Circuit. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. vol. 7417, pp. 850–867. Springer (2012)
[36] Gentry, C., Sahai, A., Waters, B.: Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013. vol. 8042, pp. 75–92. Springer (2013)
[37] Grassi, L., Rechberger, C., Rotaru, D., Scholl, P., Smart, N.P.: MPC-Friendly Symmetric Key Primitives. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. p. 430–443. ACM (2016)
[38] Halevi, S., Shoup, V.: Bootstrapping for HElib. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015. vol. 9056, pp. 641–670. Springer (2015)
[39] Han, K., Ki, D.: Better Bootstrapping for Approximate Homomorphic Encryption. In: Jarecki, S. (ed.) Topics in Cryptology – CT-RSA 2020. vol. 12006, pp. 364–390. Springer (2020)
[40] Hong, S., Lee, S., Lim, J., Sung, J., Cheon, D., Cho, I.: Provable Security against Differential and Linear Cryptanalysis for the SPN Structure. In: Goos, G., Hartmanis, J., van Leeuwen, J., Schneier, B. (eds.) Fast Software Encryption – FSE 2000. vol. 1978. Springer (2001)
[41] Jakobsen, T., Knudsen, L.R.: The interpolation attack on block ciphers. In: Biham, E. (ed.) Fast Software Encryption – FSE '97. vol. 1267, pp. 28–40. Springer (1997)
[42] Jean, J., Nikolić, I., Peyrin, T.: Tweaks and Keys for Block Ciphers: The TWEAKEY Framework. In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology – ASIACRYPT 2014. vol. 8874, pp. 274–288. Springer (2014)
[43] Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In: Proceedings of the 27th USENIX Conference on Security Symposium. p. 1651–1668. USENIX Association (2018)
[44] Kelsey, J., Schneier, B., Wagner, D.: Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In: Koblitz, N. (ed.) Advances in Cryptology – CRYPTO '96. vol. 1109, pp. 237–251. Springer (1996)
[45] Knudsen, L., Wagner, D.: Integral Cryptanalysis. In: Daemen, J., Rijmen, V. (eds.) Fast Software Encryption – FSE 2002. vol. 2365, pp. 112–127. Springer (2002)
[46] Lauter, K., López-Alt, A., Naehrig, M.: Private Computation on Encrypted Genomic Data. In: Aranha, D.F., Menezes, A. (eds.) Progress in Cryptology – LATINCRYPT 2014. vol. 8895, pp. 3–27. Springer (2015)
[47] Lepoint, T., Naehrig, M.: A Comparison of the Homomorphic Encryption Schemes FV and YASHE. In: Pointcheval, D., Vergnaud, D. (eds.) Progress in Cryptology – AFRICACRYPT 2014. vol. 8469, pp. 318–335. Springer (2014)
[48] Lyubashevsky, V., Peikert, C., Regev, O.: On Ideal Lattices and Learning with Errors over Rings. In: Gilbert, H. (ed.) Advances in Cryptology – EUROCRYPT 2010. vol. 6110, pp. 1–23. Springer (2010)
[49] Matsumoto, T., Imai, H.: Public Quadratic Polynomial-Tuples for Efficient Signature-Verification and Message-Encryption. In: Barstow, D., Brauer, W.,

Brinch Hansen, P., Gries, D., Luckham, D., Moler, C., Pnueli, A., Seegmüller, G., Stoer, J., Wirth, N., Günther, C.G. (eds.) Advances in Cryptology – EUROCRYPT '88. vol. 330, pp. 419–453. Springer (1988)
[50] Méaux, P., Journault, A., Standaert, F.X., Carlet, C.: Towards Stream Ciphers for Efficient FHE with Low-Noise Ciphertexts. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016. vol. 9665, pp. 311–343. Springer (2016)
[51] Naehrig, M., Lauter, K., Vaikuntanathan, V.: Can Homomorphic Encryption be Practical? In: Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop. p. 113–124. ACM (2011)
[52] Park, S., Byun, J., Lee, J., Cheon, J.H., Lee, J.: HE-Friendly Algorithm for Privacy-Preserving SVM Training. IEEE Access **8**, 57414–57425 (2020)
[53] Rechberger, C., Soleimany, H., Tiessen, T.: Cryptanalysis of Low-Data Instances of Full LowMCv2. IACR Transactions on Symmetric Cryptology **2018**(3), 163–181 (2018)
[54] Microsoft SEAL (release 3.4). https://github.com/Microsoft/SEAL (Oct 2019), microsoft Research, Redmond, WA.
[55] XKCP: eXtended Keccak Code Package. https://github.com/XKCP/XKCP (Aug 2020)

A Homomorphic Evaluation of Symmetric Ciphers

Homomorphic encryption can be made more efficient using batching techniques that allow to encrypt multi-dimensional arrays. Suppose that we use the FV scheme with plaintext modulus t and degree of the polynomial modulus N, and that we want to evaluate multiplication by a matrix $\mathbf{A} \in \mathbb{Z}_t^{n \times n}$ where $n|N$ and $n \ll N$. A straightforward approach is evaluating

$$\mathbf{A} \cdot \begin{pmatrix} \mathcal{C}_0 \\ \mathcal{C}_1 \\ \vdots \\ \mathcal{C}_{n-1} \end{pmatrix}$$

for encrypted arrays $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_{n-1}$ by applying homomorphic addition and multiplication. We will call this method *row-wise packing*.

Alternatively, we can evaluate

$$\begin{pmatrix} \mathbf{A} & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{A} & \ldots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{A} \end{pmatrix} \cdot \begin{pmatrix} m_0 \\ m_1 \\ \vdots \\ m_{N-1} \end{pmatrix}$$

in a single ciphertext by applying rotation as well as homomorphic operations to an encrypted array

$$\mathcal{C} = \mathsf{Enc}([m_0, \ldots, m_{N-1}]).$$

We will call this method *column-wise packing*. Not only linear layers, but also nonlinear layers can be evaluated by both of these methods.

In the CKKS-FV transciphering framework, the evaluation method should be chosen for both the client and the server sides. On the client side, for N/n pairs $\left(\mathbf{m}^{(i)}, \mathsf{E}(\mathbf{m}^{(i)})\right)_{i=0}^{N/n-1}$ of plaintext and E-ciphertext, the column-wise packing will compute

$$\mathsf{Ecd}^{\mathsf{FV}}\left(\mathsf{E}(\mathbf{m}^{(0)}), \ldots, \mathsf{E}(\mathbf{m}^{(N/n-1)})\right).$$

On the other hand, for N pairs $\left(\mathbf{m}^{(i)}, \mathsf{E}(\mathbf{m}^{(i)})\right)_{i=0}^{N-1}$, the row-wise packing will compute

$$\begin{aligned}
&\mathsf{Ecd}^{\mathsf{FV}}\left(\mathsf{E}(\mathbf{m}^{(1)})_0, \ldots, \mathsf{E}(\mathbf{m}^{(N-1)})_0\right), \\
&\mathsf{Ecd}^{\mathsf{FV}}\left(\mathsf{E}(\mathbf{m}^{(1)})_1, \ldots, \mathsf{E}(\mathbf{m}^{(N-1)})_1\right), \\
&\quad\vdots \\
&\mathsf{Ecd}^{\mathsf{FV}}\left(\mathsf{E}(\mathbf{m}^{(1)})_{n-1}, \ldots, \mathsf{E}(\mathbf{m}^{(N-1)})_{n-1}\right),
\end{aligned}$$

where $\mathsf{E}(\mathbf{m}^{(i)})_j$ implies the j-th component of $\mathsf{E}(\mathbf{m}^{(i)})$.

Supplementary Material

A The Number of Cubic Monomials in a 1-round HERA

The round function of HERA is defined by

$$\mathsf{RF} = \mathsf{ARK} \circ \mathsf{Cube} \circ \mathsf{MixRows} \circ \mathsf{MixColumns},$$

where the two linear maps MixColumns and MixRows can be represented by 16×16-matrices over $\mathbb{Z}_t$. Their product represents MixRows $\circ$ MixColumns as follows.

$$\mathsf{MixRows} \circ \mathsf{MixColumns} = \begin{pmatrix}
4 & 6 & 2 & 2 & 6 & 9 & 3 & 3 & 2 & 3 & 1 & 1 & 2 & 3 & 1 & 1 \\
2 & 4 & 6 & 2 & 3 & 6 & 9 & 3 & 1 & 2 & 3 & 1 & 1 & 2 & 3 & 1 \\
2 & 2 & 4 & 6 & 3 & 3 & 6 & 9 & 1 & 1 & 2 & 3 & 1 & 1 & 2 & 3 \\
6 & 2 & 2 & 4 & 9 & 3 & 3 & 6 & 3 & 1 & 1 & 2 & 3 & 1 & 1 & 2 \\
2 & 3 & 1 & 1 & 4 & 6 & 2 & 2 & 6 & 9 & 3 & 3 & 2 & 3 & 1 & 1 \\
1 & 2 & 3 & 1 & 2 & 4 & 6 & 2 & 3 & 6 & 9 & 3 & 1 & 2 & 3 & 1 \\
1 & 1 & 2 & 3 & 2 & 2 & 4 & 6 & 3 & 3 & 6 & 9 & 1 & 1 & 2 & 3 \\
3 & 1 & 1 & 2 & 6 & 2 & 2 & 4 & 9 & 3 & 3 & 6 & 3 & 1 & 1 & 2 \\
2 & 3 & 1 & 1 & 2 & 3 & 1 & 1 & 4 & 6 & 2 & 2 & 6 & 9 & 3 & 3 \\
1 & 2 & 3 & 1 & 1 & 2 & 3 & 1 & 2 & 4 & 6 & 2 & 3 & 6 & 9 & 3 \\
1 & 1 & 2 & 3 & 1 & 1 & 2 & 3 & 2 & 2 & 4 & 6 & 3 & 3 & 6 & 9 \\
3 & 1 & 1 & 2 & 3 & 1 & 1 & 2 & 6 & 2 & 2 & 4 & 9 & 3 & 3 & 6 \\
6 & 9 & 3 & 3 & 2 & 3 & 1 & 1 & 2 & 3 & 1 & 1 & 4 & 6 & 2 & 2 \\
3 & 6 & 9 & 3 & 1 & 2 & 3 & 1 & 1 & 2 & 3 & 1 & 2 & 4 & 6 & 2 \\
3 & 3 & 6 & 9 & 1 & 1 & 2 & 3 & 1 & 1 & 2 & 3 & 2 & 2 & 4 & 6 \\
9 & 3 & 3 & 6 & 3 & 1 & 1 & 2 & 3 & 1 & 1 & 2 & 6 & 2 & 2 & 4
\end{pmatrix}.$$

We see that the matrix representation of MixRows $\circ$ MixColumns has no zero entry. It implies that MixRows $\circ$ MixColumns contains all the linear monomials in its polynomial representation, and hence RF contains all the cubic monomials. More precisely, if $a_i \neq 0$ for $i = 0, 1, \ldots, n-1$, then we have

$$\begin{aligned}
(a_0 x_0 + a_1 x_1 + \cdots + a_{n-1} x_{n-1})^3 &= \sum_{i,j,k} a_i a_j a_k x_i x_j x_k \\
&= \sum_{i \leq j \leq k} \alpha(i,j,k) a_i a_j a_k x_i x_j x_k,
\end{aligned}$$

where

$$\alpha(i,j,k) = \begin{cases}
1 & \text{if } i = j = k; \\
3 & \text{if either } i = j < k \text{ or } i < j = k; \\
6 & \text{if } i < j < k.
\end{cases}$$

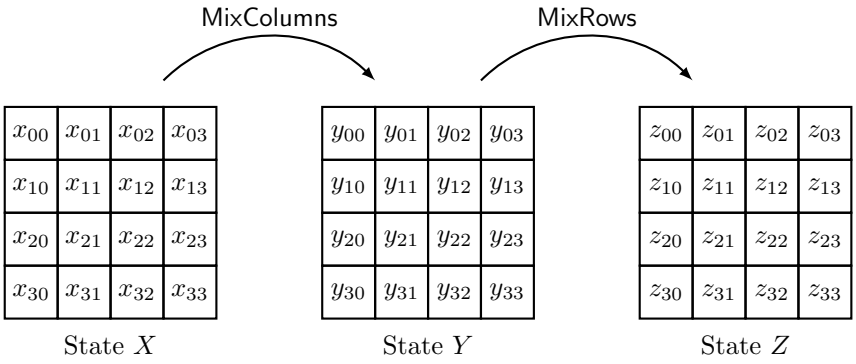


Fig. 6: Diagram of state change in HERA.

Since the plaintext modulus t is prime and $t > 6$, every monomial of degree three has a nonzero coefficient.

B Branch Number of the Linear Layer in HERA

In this section, we compute the branch number of the linear layer of HERA. Given a square matrix $\mathbf{M}$ over a finite field, its linear branch number B_ℓ and differential branch number B_d are defined by

$$B_\ell(\mathbf{M}) = \min_{\mathbf{x}\neq 0}\{\mathsf{hw}(\mathbf{x}) + \mathsf{hw}(\mathbf{M}^T\mathbf{x})\},$$
$$B_d(\mathbf{M}) = \min_{\mathbf{x}\neq 0}\{\mathsf{hw}(\mathbf{x}) + \mathsf{hw}(\mathbf{M}\mathbf{x})\},$$

respectively, where hw denotes the word-wise hamming weight function. For example, the differential branch number of the 4×4 submatrix

$$\mathbf{L} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \tag{2}$$

used in MixColumns is 5, which means that at least five nonzero components (active S-boxes) exist in an input and the output of $\mathbf{L}$. It is easily seen that $2 \leq B_\ell(\mathbf{M}), B_d(\mathbf{M}) \leq n + 1$ for an $n \times n$-matrix $\mathbf{M}$. It has also been proved that $B_\ell(\mathbf{M}) = n + 1$ if and only if $B_d(\mathbf{M}) = n + 1$ [40]. A matrix $\mathbf{M}$ such that $B_\ell(\mathbf{M}) = B_d(\mathbf{M}) = n+1$ is called a *maximum distance separable* (MDS) matrix.

One can computationally prove that $\mathbf{L}$ is an MDS matrix over $\mathbb{Z}_t$ when t is prime and $t > 17$. It implies that the linear and the differential branch numbers of MixColumns and MixRows are all 5.

Theorem 2. *The linear and the differential branch numbers of*

$$\mathsf{MixRows} \circ \mathsf{MixColumns}$$

are all 8.

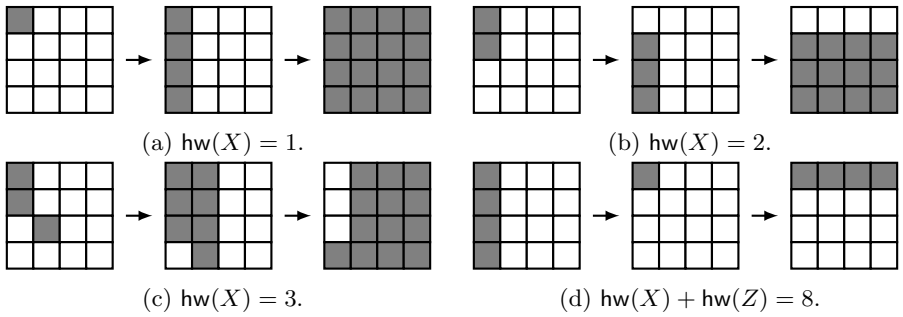
Proof. We will prove that the differential branch number of MixRows∘MixColumns is 8. The linear branch number is computed similarly. We use the notations in Figure 6.

Suppose that the branch number B_d of $\mathsf{MixRows} \circ \mathsf{MixColumns}$ is less than 8. It means that there exists a triple of nonzero states (X, Y, Z) such that $\mathsf{hw}(X) + \mathsf{hw}(Z) \leq 7$. Assuming $\mathsf{hw}(X) \leq 3$, we distinguish the following three cases.

- $\mathsf{hw}(X) = 1$: all the components of Z are nonzero as seen in Figure 7a.
- $\mathsf{hw}(X) = 2$: two nonzero components might be in the same column or not. For either case, $\mathsf{hw}(Z) \geq 12$ (see Figure 7b for the first subcase).
- $\mathsf{hw}(X) = 3$: we need to consider three subcases: three nonzero components are in the same column, only two are in the same column, or all three nonzero components are in different columns. $\mathsf{hw}(Z) \geq 8$ for the first subcase, $\mathsf{hw}(Z) \geq 13$ for the second subcase, and $\mathsf{hw}(Z) \geq 8$ for the third subcase (see Figure 7c for the second subcase).

Next, we assume that $\mathsf{hw}(X) \geq 4$; it implies $\mathsf{hw}(Z) \leq 3$. By the symmetry between MixColumns and MixRows, it should be possible to draw a state change diagram (like Figure 7a, 7b, 7c) such that $\mathsf{hw}(X) \leq 3$ and $\mathsf{hw}(X) + \mathsf{hw}(Z) \leq 7$ if there is a triple of states (X, Y, Z) such that $\mathsf{hw}(Z) \leq 3$ and $\mathsf{hw}(X) + \mathsf{hw}(Z) \leq 7$. So, there is no triple of states (X, Y, Z) such that $\mathsf{hw}(X) + \mathsf{hw}(Z) \leq 7$.

Finally, we completes the proof by giving an example satisfying $\mathsf{hw}(X) + \mathsf{hw}(Z) = 8$. See Figure 7d. A specific example of Figure 7d can be obtained by fixing an intermediate state Y such that $\mathsf{hw}(Y) = 1$, and then applying $\mathsf{MixColumns}^{-1}$ and MixRows, respectively, to Y. □



(a) $\mathsf{hw}(X) = 1$. (b) $\mathsf{hw}(X) = 2$.

(c) $\mathsf{hw}(X) = 3$. (d) $\mathsf{hw}(X) + \mathsf{hw}(Z) = 8$.

Fig. 7: Pictorial representation of four cases appearing in the proof of Theorem 2. A gray-colored cell represents a nonzero component.

C Bounding the Encoding Error

When a client encodes a message $\mathbf{m} \in \mathbb{C}^{\ell/2}$ within the CKKS-FV framework, a small error inevitably occurs. For $M = \mathsf{Ecd}^{\mathsf{CKKS}}(\mathbf{m}, \mathrm{scale} = \delta)$, it is easy to see

that the encoding error

$$\|\mathsf{Dcd}^{\mathsf{CKKS}}(M, \text{scale} = \delta) - \mathbf{m}\|_\infty$$

is upper bounded by $\frac{\ell}{2\delta}$. However, for smaller ciphertext expansion, we prove a slightly sharper bound as follows.

Lemma 2. *Let δ be a positive number such that $\delta \geq 2$. Then for*

$$M = \mathsf{Ecd}^{\mathsf{CKKS}}(\mathbf{m}, \text{scale} = \delta),$$

we have

$$\|\mathsf{Dcd}^{\mathsf{CKKS}}(M, \text{scale} = \delta) - \mathbf{m}\|_\infty < \frac{\alpha\ell}{\delta},$$

where $\alpha < 0.476$.

Proof. Let $M' \in \mathbb{Q}[X]/(\Phi_{2\ell}(X))$ be the CKKS-encoded polynomial with scaling factor δ but not rounded off to integer coefficients. Then,

$$\|M - M'\|_\infty < \frac{1}{2}.$$

For a 2ℓ-th primitive root ζ, we have

$$\begin{aligned}
|M(\zeta) - M'(\zeta)| &\leq \max_{\substack{a_i \in \mathbb{R} \\ |a_i| \leq 1/2}} |a_0 + a_1\zeta + \cdots + a_{\ell-1}\zeta^{\ell-1}| \\
&= \max_{\substack{a_i \in \mathbb{R} \\ |a_i| \leq 1/2}} \left(\left(\sum_{j=0}^{\ell-1} a_j \cos(j\pi/\ell) \right)^2 + \left(\sum_{j=0}^{\ell-1} a_j \sin(j\pi/\ell) \right)^2 \right)^{\frac{1}{2}} \\
&\leq \frac{1}{2} \left(\left(\sum_{j=0}^{\ell-1} |\cos(j\pi/\ell)| \right)^2 + \left(\sum_{j=0}^{\ell-1} |\sin(j\pi/\ell)| \right)^2 \right)^{\frac{1}{2}},
\end{aligned}$$

where $\sin(j\pi/\ell)$ is non-negative for every $j = 0, 1, \ldots, \ell - 1$, so their sum can be estimated by using the Taylor series as follows.

$$\sum_{j=0}^{\ell-1} |\sin(j\pi/\ell)| \lesssim -0.262 + 0.704 \cdot \ell + O(1/\ell). \tag{3}$$

We also have

$$\sum_{j=0}^{\ell-1} |\cos(j\pi/\ell)| = \sum_{j=0}^{\ell/2-1} (\cos(j\pi/\ell) + \sin(j\pi/\ell)).$$

Hence it also can be estimated by using the Taylor series as follows.

$$\sum_{j=0}^{\ell-1} |\cos(j\pi/\ell)| \lesssim -0.012 + 0.638 \cdot \ell + O(1/\ell). \tag{4}$$

By (3) and (4), we have

$$|M(\zeta) - M'(\zeta)| \lesssim 0.476 \cdot \ell,$$

which completes the proof. □