Hybrid Framework for Approximate Computation over Encrypted Data

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

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

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 computational overload and the ciphertext expansion, we propose a novel hybrid framework that supports CKKS. Since it seems 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-friendly ciphers such as FLIP and Rasta using randomized linear layers, HERA needs a 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 2.386 to 230.0 times faster on the client-side, compared to the environment where only CKKS is used, in terms of encryption time. Our framework also enjoys 2.357 to 180.7 times smaller ciphertext expansion according to the plaintext length. In the transciphering framework with BGV only, HERA increases the throughput on the client side (resp. server side) up to 229380 times (resp. 355478 times) compared to existing HE-friendly ciphers.

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 environments. 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 [31], there has been a large amount of research in this area [13, 29, 23, 33]. Various applications of HE to medical, genomic, and financial tasks have also been proposed [20, 22, 43, 49].

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 [12, 50].

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 [32], 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 [49] (see Figure 1). In the client-sever model, a client encrypts a message $\mathbf{m}$ using a symmetric cipher E with a

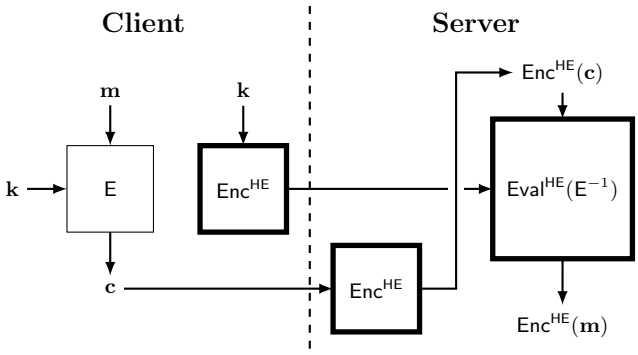


Fig. 1: The (basic) transciphering framework. Homomorphic operations are performed in the boxes with thick lines.

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.

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 [48] and Rasta [25].

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.

1.1 Our Contribution

The main contribution 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. The problem is that it seems infeasible to design a symmetric cipher operating on real numbers. In order to overcome this problem, we combine CKKS with FV which is a homomorphic encryption scheme using modular arithmetic [29], 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 (HE-friendly cipher with an RAndomized key schedule), 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 any existing construction 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	p	Ciphertext Expansion			Performance	
				Msg.	Ctxt.	Ratio	Latency	Throughput
Short	CKKS-FV	-	9	18 B	34 B	1.89	1.645 μs	10689 KB/s
	CKKS (level 0)	2^{10}			6144 B	341.3	378.1 μs	46.48 KB/s
	CKKS (full level)	2^{10}			6912 B	384	378.1 μs	46.48 KB/s
Long	CKKS-FV	-	10	40 KB	112 KB	2.8	4.801 ms	8.136 MB/s
	CKKS (level 0)	2^{15}			264 KB	6.6	11.46 ms	3.409 MB/s
	CKKS (full level)	2^{15}			6432 KB	160.8	170.1 ms	0.2296 MB/s

Table 1: Comparison of the CKKS-FV transciphering framework with HERA and the CKKS-only environment. All the experiments are done with 128-bit security. The parameter p stands for the bits of precision.

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, 3, 4, 34], 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 requires too many random bits in the generation of 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.

Implementation. 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 is only 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 is set to 128. 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 2.386 to 230.0 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 2.357 to 180.7 times smaller than CKKS only (of level 0).

In order to make a fair comparison of HERA to existing HE-friendly ciphers, we also implemented them in the transciphering framework using the BGV scheme in the `HElib` library [38]. The comparison is given in Table 2 (and in Section 6.2 in detail). We see that HERA increases the throughput on the client side (resp. server side) 1.250 to 229380 times (resp. 6.235 to 355478 times) compared to existing HE-friendly ciphers.

Cipher(Parameters)	Client-side		Server-side	
	Latency (cycles)	Throughput (C/B)	Latency (s)	Throughput (KB/s)
80-bit				
LowMCv3$(13, 128, 31)$	6.118×10^4	3824	332.8	3.380×10^{-2}
FLIP$(42, 128, {}^8\Delta^9)$	$\geq 4.303 \times 10^6$	$\geq 3.443 \times 10^7$	98.49	1.004×10^{-4}
Rasta$(4, 327, 2)$	$\geq 2.224 \times 10^7$	$\geq 5.440 \times 10^5$	265.7	3.005×10^{-4}
Dasta$(4, 327, 2)$	8.648×10^4	2116	-	-
Masta$(5, 16, 65537)$	6004	187.6	57.95	1.104
HERA$(4, 16, 65537)$	4803	150.1	28.69	35.69
128-bit				
LowMCv3$(14, 196, 63)$	1.496×10^5	6106	1191	3.615×10^{-2}
FLIP$(82, 224, {}^8\Delta^{16})$	$\geq 1.080 \times 10^7$	$\geq 8.643 \times 10^7$	1195	1.532×10^{-5}
Rasta$(5, 525, 2)$	$\geq 7.393 \times 10^7$	$\geq 1.127 \times 10^6$	1517	8.449×10^{-5}
Dasta$(5, 525, 2)$	1.538×10^5	2344	-	-
Masta$(6, 32, 65537)$	1.373×10^4	214.5	768.7	2.664
HERA$(5, 16, 65537)$	4911	153.5	36.64	27.95

Table 2: Comparison of HERA to existing HE-friendly ciphers with BGV.

1.2 Related Work

Since the transciphering framework has been introduced [49], early works have been focused on homomorphic evaluation of popular symmetric ciphers (e.g., AES [32], SIMON [44], and PRINCE [27]). Such ciphers have been designed without any consideration on their arithmetic complexity, so the performance of their homomorphic evaluation was not satisfactory. In this line of research, LowMC [2] is the first construction that aims to minimize the depth and the number of AND gates. However, it turned out that LowMC is vulnerable to algebraic attacks [24, 26, 51], so it has been revised later.[3]

Canteaut et al. [14] claimed that stream ciphers would be advantageous in terms of online complexity compared to block ciphers, and proposed a new stream cipher Kreyvium. However, its practical relevance is limited since the multiplicative depth (with respect to the secret key) keeps growing as keystreams are generated. The FLIP stream cipher [48] is based on a novel design strategy that its permutation layer is randomly generated for every encryption without increasing the algebraic degree in its secret key. Furthermore, it has been reported that FiLIP [47], a generalized instantiation of FLIP, can be efficiently evaluated with the TFHE scheme [39]. Rasta [25] is a stream cipher aiming at higher throughput at the cost of high latency using random linear layers, which are generated by an extendable output function. Dasta [37], a variant of Rasta using affine layers with lower entropy, boosts up the client-side computation. As another variant of Rasta, Masta [35] operates on a modular domain, improving upon Rasta in terms of the throughput of homomorphic evaluation.

Beside the transciphering framework and HE-friendly ciphers, there have been attempts to reduce the memory overhead when encrypting short messages. Chen et al. [17] proposed a conversion method between LWE ciphertexts and RLWE ciphertexts. Small messages can be encrypted by LWE-based symmetric encryption with small ciphertext expansion, and a collection of LWE ciphertexts is converted to an RLWE ciphertext to perform a homomorphic evaluation. Chen et al. [18] proposed a hybrid HE scheme using the CKKS packing algorithm and a variant of FV. 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.

[3] `https://github.com/LowMC/lowmc/blob/master/determine_rounds.py`

For a complex vector $\mathbf{x}$, its ℓ_p-norm is denoted by $\|\mathbf{x}\|_p$. Usual dot products of vectors are denoted by $\langle \cdot, \cdot \rangle$. 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. We denote the multiplicative group of $\mathbb{Z}_t$ by $\mathbb{Z}_t^\times$. The set of strings of arbitrary length over a set S is denoted by S^*. For two vectors (strings) $\mathbf{a}$ and $\mathbf{b}$, their concatenation is denoted by $\mathbf{a}\|\mathbf{b}$. 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 of which security is based on the hardness of Ring Learning With Errors (RLWE) problem [52, 45]. For more details, we refer to [29, 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* computation over the real numbers by taking different strategies to efficiently encode messages.

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 $0 \le i \le N-1$.[4]

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 $0 \le j \le N/2 - 1$.

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

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 $q_0, \ldots, q_L$ such that q_i divides q_{i+1} for $0 \leq i \leq L-1$, 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 $e_0, e_1 \leftarrow \mathcal{D}_{err}$.
  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

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

  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.

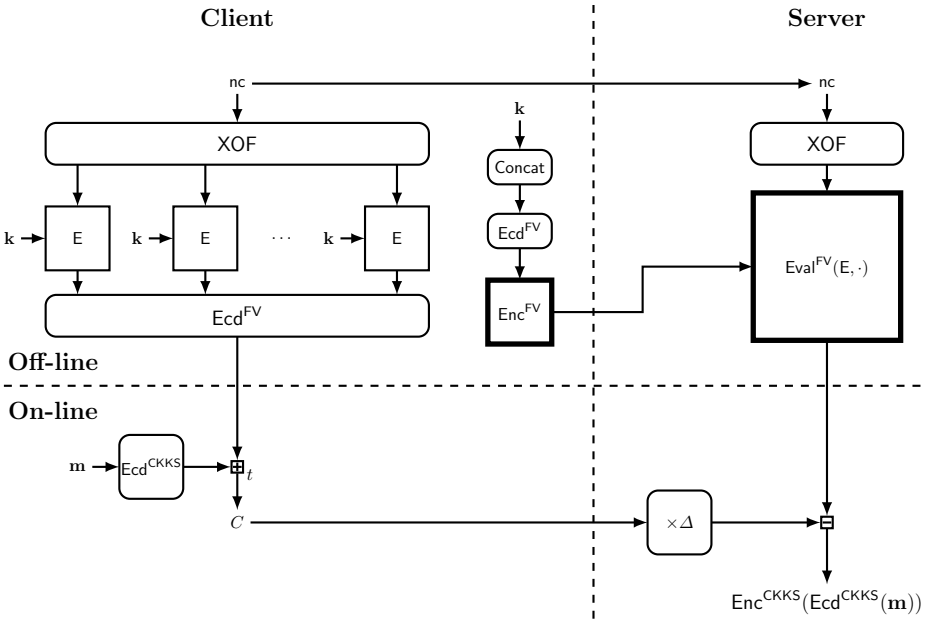


Fig. 2: The CKKS-FV transciphering framework. Homomorphic encryption and evaluation is performed in the boxes with thick lines. Operations in the boxes with rounded corners do not use any secret information. The vertical dashed line distinguishes the client-side and the server-side computation, while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

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 by $\mathsf{E}_{\mathbf{u}}$ (or simply E).

Generation of an instance of E by $\mathbf{u} \in \mathbb{Z}_t^*$ is denoted by Gen (resp. Gen′) in Algorithm 1 (resp. Algorithm 2). The input $\mathbf{u}$ is again generated by the underlying extendable output function $\mathsf{XOF} : \{0,1\}^\lambda \times \mathbb{Z} \to \mathbb{Z}_t^*$ 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 $\mathbf{k}$, (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-ciphertexts 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

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

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. The client computes

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

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 by $\mathsf{Eval}^{\mathsf{FV}}(\mathsf{E}, \cdot)$. The procedure of generating the stream cipher and constructing a circuit for $\mathsf{Eval}^{\mathsf{FV}}(\mathsf{E}, \cdot)$ is denoted by 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.

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 $\mathcal{M}$

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 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 $\|e_{\mathsf{eval}}\|_\infty < \Delta/2$ (i.e., the ciphertext is correctly FV-decryptable), then we have*

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

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

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

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

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

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

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

where $\|\cdot\|_\infty^{can} \coloneqq \|\mathsf{Dcd}^{\mathsf{CKKS}}(\cdot, \mathrm{scale} = 1)\|_\infty$. □

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

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

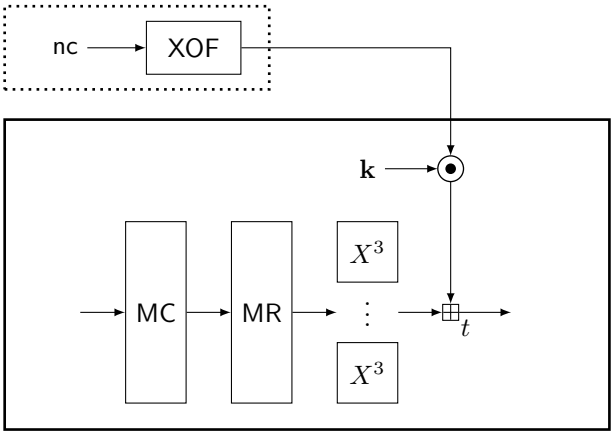


Fig. 3: The round function of HERA. Operations in the box with dotted (resp. thick) lines are public (resp. secret). "MC" and "MR" represent MixColumns and MixRows, respectively.

for $i = 0, \ldots, r$, and $\mathbf{x} \in \mathbb{Z}_t^{16}$, where $\bullet$ (resp. $+$) denotes component-wise multiplication (resp. addition) modulo t. The extendable output function XOF might be instantiated with a sponge-type hash function SHAKE256 [28].

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 $\mathbb{Z}_t$.

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

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

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

(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.

For the bijectivity of S-boxes, it is required that $\gcd(3, t-1) = 1$.

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

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

where $\mathbf{ic}$ denotes a constant $(1, 2, \ldots, 16) \in \mathbb{Z}_t^{16}$.

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., FLIP [48] and Rasta [25]). 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 literally extending FLIP [48] or Rasta [25] to 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 3, 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 16×16 square matrices 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 3: Comparisons of different types of 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 (Maximum Distance Separable) 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, which makes algebraic attacks infeasible. Also, using MixRows mitigates linear cryptanalysis; the branch number of the linear layer is 8 (See Appendix C) so that HERA does not have a high-probability linear trail.

In the nonlinear layer, Cube takes the component-wise cube of the input. The cube map is studied from earlier multivariate cryptography [46], recently attracting renewed interest for the use in MPC/ZKP-friendly ciphers [1, 3]. 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 ciphers such as FLIP and Rasta.

To balance between efficiency and 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 cannot obtain this effect since an adversary is able to use the linear change of variables (see Section 5.1). Based on this observation, we opt for component-wise multiplication. It offers better security on algebraic attacks and linear cryptanalysis.

The input constant $\mathbf{ic} = (1, 2, \ldots, 16)$ consists of distinct numbers in $\mathbb{Z}_t^{16}$; it will make a larger number of monomials in the polynomial representation of the cipher (compared to using a too simple constant, say the all-zero vector), enhancing security against algebraic attacks.

5 Security Analysis of HERA

In this section, we provide security analyses of HERA. Table 4 shows the number of rounds to prevent each of the attacks considered in this section according to the security level λ, where we assume that $t > 2^{16}$.

Attack \ λ	80	128	192	256
Trivial Linearization	4	5	6	7
GCD Attack	1	1	1	7
Gröbner Basis Attack	4	5	6	7
Interpolation Attack	4	5	6	7
Linear Cryptanalysis	2	4	4	6

Table 4: Recommended number of rounds with respect to each attack.

Assumptions and the Scope of Analysis. We limit the number of encryptions under the same key up to the birthday bound with respect to λ, i.e., $2^{\lambda/2}$, since otherwise one would not be able to avoid a nonce collision (when nonces are chosen uniformly at random).

In this work, we will consider the standard "secret-key model", where an adversary arbitrarily chooses a nonce, and obtains the corresponding keystream without any information on the secret key. The related-key and the known-key models are beyond the scope of this paper.

Since HERA takes as input counters, an adversary is not able to control the differences of the inputs. Nonces can be adversarially chosen, while they are also fed to the extendable output function, which is modeled as a random oracle. So one cannot control the difference of the internal variables. For this reason, we believe that our construction is secure against any type of chosen-plaintext attack including (higher-order) differential, truncated differential, invariant subspace trail and cube attacks. A recent generalization of an integral attack [9] requires only a small number of chosen plaintexts, while it is not applicable to HERA within the birthday bound.

The HERA cipher can be represented by a set of polynomials over $\mathbb{Z}_t$ in unknowns $k_0, \ldots, k_{15}$, where $k_i \in \mathbb{Z}_t$ denotes the i-th component of the secret key $\mathbf{k} \in \mathbb{Z}_t^{16}$. 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. With this observation, our analysis will be focused on algebraic attacks.

5.1 Trivial Linearization

Trivial linearization is to make a system of polynomial equations *linear* by replacing all monomials by new variables, and solve it. 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. In Appendix B, we explain the reason that all the monomials of degree $\leq 3^r$ are expected to appear after r rounds of HERA. Therefore, we can conclude that 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 $O(t^g S_g^\omega)$.

Parameters. With respect to the trivial linearization attack, the recommended number of rounds is given in Table 5 for various security levels. It has been computed by the above estimation for S^ω with $\omega = 2$. Guessing variables will not affect the security of HERA when $t > 2^{16}$.

On the First and the Last Affine Layers. A classic substitution-permutation block cipher does not begin with nor end with affine layers since they do not

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

Table 5: Recommended number of rounds with respect to trivial linearization.

affect the overall security of the cipher. However, this is not the case for HERA. In the following, we give a trivial linearization attack for a variant of HERA where either the first or the last affine layer is missing.

If the first affine layer is missing, then the output of the first nonlinear layer will be of the form

$$(a_0(k_0+1)^3, a_1(k_1+2)^3, \ldots, a_{15}(k_{15}+16)^3)$$

for public coefficients $a_0, \ldots, a_{15}$, and unknowns (key variables) $k_0, \ldots, k_{15}$. An adversary can try trivial linearization with respect to the variables $(k'_0, \ldots, k'_{15})$ where $k'_i = (k_i+i+1)^3$ for $i = 0, \ldots, 15$. The maximum number of all monomials is

$$S = \sum_{i=0}^{3^{r-1}} \binom{16+i-1}{i}$$

which is much lower than expected with the first affine layer.

On the other hand, if the last affine layer is missing, the input to the last nonlinear layer will be of the form

$$((c_0 - a_0k_0)^\alpha, (c_1 - a_1k_1)^\alpha, \ldots, (c_{15} - a_{15}k_{15})^\alpha)$$

where α is the inverse of 3 modulo $(t-1)$, $c = (c_0, \ldots, c_{15})$ is a keystream, and a_i is the i-th component of $\mathbf{rc}_r$. It can be converted to the form

$$(a_0^\alpha(b_0 - k_0)^\alpha, a_1^\alpha(b_1 - k_1)^\alpha, \ldots, a_{15}^\alpha(b_{15} - k_{15})^\alpha)$$

where $b_i = c_i a_i^{-\alpha}$. Setting new variables $k_{i,j} = k_i - j$ for $0 \le j \le t-1$, for r-round HERA^r, we can establish a meet-in-the-middle (MitM) equation

$$\mathsf{HERA}^{r-1}[k, \mathsf{nc}] = (-a_i^\alpha k_{i,b_i}^\alpha)_{0 \le i \le 15}.$$

When the number of equations is sufficiently large, the maximum number of monomials is given as

$$S = 16(t-1) + \sum_{i=0}^{3^{r-1}} \binom{16+i-1}{i}$$

which is also much lower than expected with the last affine layer.

5.2 GCD Attack

The GCD attack is one of the major threats to symmetric ciphers over large fields (e.g., MiMC [1]). Let $E : \mathbb{F}_q \times \mathbb{F}_q \to \mathbb{F}_q$ be an encryption function, where $\mathbb{F}_q$ is the key and the plaintext/ciphertext spaces. Given two plaintext-ciphertext pairs (x_1, y_1), $(x_2, y_2) \in \mathbb{F}_q^2$, one can establish univariate polynomial equations $E(K, x_1) = y_1$ and $E(K, x_2) = y_2$ in an unknown K. Then,

$$\gcd(E(K, x_1) - y_1, E(K, x_2) - y_2)$$

includes $(K - k)$ as a factor, where k denotes the actual secret key. It is known that the complexity of finding the GCD of two univariate polynomials of degree d is $O(d \log^2 d)$.

This attack can be extended to a system of multivariate polynomial equations by guessing all the key variables except one. For r-round HERA, the complexity of the GCD attack is estimated as $O(t^{15} r^2 3^r)$. For a security parameter $\lambda \leq 240$, HERA will be secure against the GCD attack even with a single round as long as $t > 2^{16}$. If $\lambda = 256$, then the number of round should be at least 7.

5.3 Gröbner Basis Attack

The Gröbner basis attack is an attack by solving 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 after carefully converting the order of monomials. We refer to [3] for details. In the literature, security against Gröbner basis attack is bounded by the time complexity for Gröbner basis computing.

Suppose that an attacker wants to solve a system of m polynomial equations in n variables over a field $\mathbb{F}_q$,

$$f_1(x_1, \ldots, x_n) = f_2(x_1, \ldots, x_n) = \cdots = f_m(x_1, \ldots, x_n) = 0.$$

The complexity of computing a Gröbner basis of such system is known to be

$$O\left(\binom{n + d_{reg}}{d_{reg}}^{\omega}\right)$$

in terms of the number of operations over the base field, where $2 \leq \omega \leq 3$ is the linear algebra constant and d_{reg} is the *degree of regularity* [8]. With the degree of regularity, one can see how much degree of polynomial multiples will be needed to find the Gröbner basis. Unfortunately, it is hard to compute the exact degree of regularity for a generic system of equations. On the other hand, there is the Macaulay bound

$$d_{reg} = 1 + \sum_{i=1}^{m} (d_i - 1)$$

for a *regular sequence* (with $m = n$) where d_i is the degree of f_i [6]. When the number of equations is larger than the number of variables, the degree of

regularity of a *semi-regular sequence* can be computed as the degree of the first non-positive coefficient in the Hilbert series

$$\mathrm{HS}(z) = \frac{1}{(1-z)^n} \times \prod_{i=1}^{m}(1-z^{d_i}).$$

As it is conjectured that most sequences are semi-regular [30], we analyze the security of HERA against the Gröbner basis attack under the (semi-)regular assumption.

Hybrid Approach. One can take a hybrid approach between the guess-and-determine attack and the algebraic attack [7]. Guessing some variables makes the system of equations overdetermined. An overdetermined system becomes easier to solve; the complexity of the hybrid approach after g guesses is given as

$$O\left(q^g \binom{n-g+d_g}{d_g}^{\omega}\right)$$

where d_g is the degree of regularity after g guesses.

Application to HERA. For the Gröbner basis attack, re-arranging equations may lead to a significant impact on the attack complexity. For example, one may set a system of equations using only plaintext-ciphertext pairs, or set an equation with new variables standing for internal states. The former will be a higher-degree system in a fewer variables, while the latter will be a lower-degree system in more variables.

From a set of nonce-plaintext-ciphertext triples $\{(\mathsf{nc}_i, \mathbf{m}_i, \mathbf{c}_i)\}$, an attacker will be able to establish an over-determined system of equation

$$f_1(k_0, \ldots, k_{15}) = f_2(k_0, \ldots, k_{15}) = \cdots = f_m(k_0, \ldots, k_{15}) = 0$$

where $k_i \in \mathbb{Z}_t$ is i-th component of the key variable. The degree of regularity of the system is computed as the degree of the first non-positive coefficient in

$$\left(1-z^{3^r}\right)^{m-16}\left(\sum_{i=0}^{3^r-1} z^i\right)^{16}$$

where r is the number of rounds. Since the summation does not have any negative term, one easily see that the degree d_{reg} of regularity cannot be smaller than 3^r. So, in this case, the time complexity is lower bounded by

$$O\left(\binom{16+3^r}{3^r}^2\right).$$

Note that the hybrid approach has worse complexity when $r \leq 6$ and $t > 2^{16}$. Even for $r = 7$, there is no significant impact on the security.

Instead of a system of equations of degree 3^r, one can establish a system of $16rk$ cubic equations in $16(r-1)k+16$ variables, where k is the block length of each query. Then, the complexity is

$$O\left(\binom{16(r-1)k+16+d_{reg}(r,k)}{d_{reg}(r,k)}^{\omega}\right).$$

We compute the degree $d_{reg}(r,k)$ of regularity assuming semi-regular sequences in Table 6, and give a plot of complexity according to k in Figure 6a. We see that the degree of regularity increases from $k=2$ as k grows. It seems too costly to attack HERA with this method.

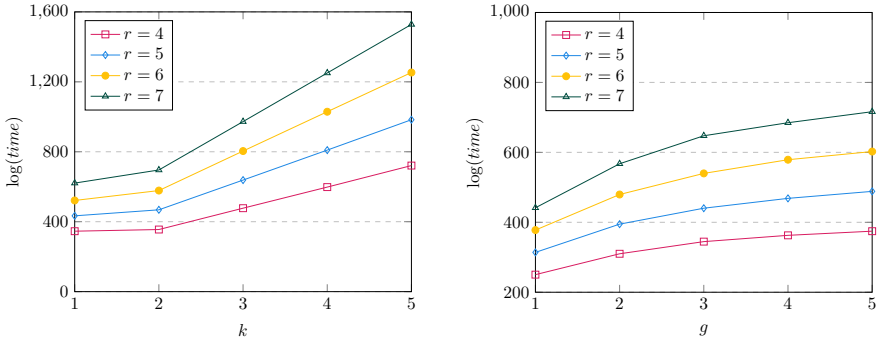
r \ k	1	2	3	4	5
4	129	75	95	115	136
5	161	100	129	159	190
6	193	124	165	206	247
7	235	151	202	254	306

Table 6: $d_{reg}(r,k)$ for a semi-regular system of $16rk$ cubic equations in $16(r-1)k+16$ variables.

r \ g	1	2	3	4	5
4	65	59	55	51	48
5	81	74	69	65	62
6	97	89	84	80	76
7	113	105	101	94	90

Table 7: $d'_{reg}(r,g)$ for a semi-regular system of $16r$ cubic equations in $16r$ variables with g guesses.

Suppose that one takes the hybrid approach. As the degree of regularity differs with a huge gap between $m=n$ and $m=n+1$, we arrange the degrees $d'_{reg}(r,g)$ of regularity when $k=1$ and the number g of guess is positive in Table 7, and give a plot of complexity depending g in Figure 6b. Provided that the



(a) Complexity as a function in k. (b) Complexity as a function in g.

Fig. 6: Complexity of the Gröbner basis attack with $\omega = 2$ and $t = 2^{16} + 1$.

complexity is

$$O\left(t^g \binom{16r - g + d'_{reg}(r,g)}{d'_{reg}(r,g)}^{\omega}\right),$$

we see that HERA provides at least 256-bit level of security for any set of parameters.

Parameters. With respect to the Gröbner basis attack, the recommended number of rounds is given in Table 8 for a various security level. It has been computed by the above estimation for $\binom{16+3^r}{3^r}^2$. Guessing variables will not affect the security of HERA when $t > 2^{16}$.

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

Table 8: Recommended number of rounds with respect to the Gröbner basis attack.

5.4 Interpolation Attack

The interpolation attack is to establish an encryption polynomial in plaintext variables without any information on the secret key and to distinguish it from a random permutation [41].

Suppose that encryption with a fixed key can be represented as a permutation on $\mathbb{F}_q^n$ (i.e., the plaintext and the ciphertext spaces are $\mathbb{F}_q^n$). Then a single word

of the ciphertext can be written as a function, say $E : \mathbb{F}_q^n \to \mathbb{F}_q$. We can also consider its polynomial representation as follows.

$$E(\mathbf{X}) = \sum_{\mathbf{e} \in \mathcal{I}} a_{\mathbf{e}} \mathbf{X}^{\mathbf{e}},$$

where $\mathcal{I} \subset \mathbb{Z}_{\geq 0}^n$ denotes the index set, and $\mathbf{X}^{\mathbf{e}} = \prod_{i=1}^n X_i^{e_i}$ for $\mathbf{X} = (X_1, \ldots, X_n) \in \mathbb{F}_q^n$ and $\mathbf{e} = (e_1, \ldots, e_n) \in \mathcal{I}$. Given plaintext-ciphertext pairs $(\mathbf{X}_j, Y_j)_j$ such that $Y_j = E(\mathbf{X}_j)$, this attack seeks to recover all the (key-related) coefficients $a_{\mathbf{e}}$'s i.e., a polynomial equivalent to E. It is known that the data complexity of this attack depends on the number of monomials in the polynomial representation of this cipher.

For the r-round HERA cipher, let $\mathbf{rc} = (\mathbf{rc}_0, \ldots, \mathbf{rc}_r) \in (\mathbb{Z}_t^{16})^{r+1}$ be a sequence of the outputs from XOF. For $i = 0, \ldots, r$, $\mathbf{rc}_i$ is evaluated by a polynomial of degree 3^{r-i}. As we expect that the r-round HERA cipher has almost all monomials of degree $\leq 3^r$ in its polynomial representation, the number of monomials is lower bounded by

$$\sum_{j=0}^{r} \sum_{i=0}^{3^j} \binom{16+i-1}{i}.$$

Considering the data limit up to the birthday bound, the recommended number of rounds with respect to this attack is summarized in Table 9.

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

Table 9: Recommended number of rounds with respect to the interpolation attack.

One might try to recover the secret key using the interpolation attack on $r-1$ rounds. However, HERA uses the full key material for every round. It implies that the key recovery attack needs brute-force search for all the key space.

Meet-in-the-middle Approach. In the basic approach, one considers a polynomial equivalent to the output, while one might try to find a polynomial equation in the middle state. However, the inverse of the cube map is of degree $(2t-1)/3$, so the degree of the equation in the middle state will be too high to recover all its coefficients. So we conclude that the meet-in-the-middle approach is not applicable to HERA.

5.5 Linear Cryptanalysis

Linear Cryptanalysis on Non-binary Ciphers. Linear cryptanalysis is typically applied to block ciphers operating on binary spaces. However, linear

cryptanalysis can be extended to non-binary spaces [5]; similarly to binary ciphers, for a prime t, the linear probability of a cipher $\mathsf{E} : \mathbb{Z}_t^n \to \mathbb{Z}_t^n$ with respect to input and output masks $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_t^n$ 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 $\mathbb{Z}_t^n$. When E is a random permutation, the expected linear probability is defined by

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

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 $\max_{\mathbf{a} \neq 0, \mathbf{b}} \mathrm{ELP}^{\mathsf{E}}(\mathbf{a}, \mathbf{b})$.

Application to HERA. One might consider two different approaches in the application of linear cryptanalysis on HERA according to how to take the input variables: the XOF output variables or the key variables. In the first case, unlike traditional linear cryptanalysis, the probability of any linear trail of HERA depends on the key since it is multiplied to the input. It seems infeasible to make a plausible linear trail without any information on the key material.

In the second case, the attack is reduced to solving an LWE-like problem as follows; given pairs $(\mathsf{nc}_i, \mathbf{y}_i)$ such that $\mathsf{HERA}(\mathbf{k}, \mathsf{nc}_i) = \mathbf{y}_i$, one can establish

$$\langle \mathbf{b}, \mathbf{y}_i \rangle = \langle \mathbf{a}, \mathbf{k} \rangle + e_i$$

for some vectors $\mathbf{a} \neq \mathbf{0}, \mathbf{b} \in \mathbb{Z}_t^n$ and error e_i sampled according to a certain distribution χ. It requires $1/\mathrm{ELP}^{\mathsf{E}}(\mathbf{a}, \mathbf{b})$ samples to distinguish χ from the uniform distribution [5]. The linear probability of 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 (linear) 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 [15] proved that

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

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

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

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

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

Parameters. Based on the evaluation given as above, the recommended number of rounds is summarized in Table 10 for various security levels. This table assumes that $t > 2^{16}$.

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

Table 10: Recommended number of rounds with respect to linear cryptanalysis.

5.6 Differential Cryptanalysis

Although we believe that our construction will be secure against any type of chosen-plaintext attack, for an unsuspected differential-related attack, we present a computation of differential characteristic in the following. Resistance of a substitution-permutation cipher against differential cryptanalysis is typically estimated by the maximum probability of differential trails [10]. Let $S : \mathbb{Z}_t \to \mathbb{Z}_t$ be the nonlinear map (S-box) used in Cube. Given a pair $(\alpha,\beta) \in \mathbb{Z}_t^\times \times \mathbb{Z}_t$, the differential probability of S is defined by

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

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 Appendix C), an r-round HERA cipher provides λ-bit security against differential cryptanalysis when

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

Since this inequality is similar to one for linear cryptanalysis, the required number of rounds is same as in Table 10.

6 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 [53] 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 compiler in O3 optimization level. XOF is instantiated with SHAKE256 in `XKCP` [54]. In the client-side implementation, we use the AVX2 instruction set.

We also evaluate the performance of HERA combined with BGV only in order to make a fair comparison with previous works. Our source codes are developed using `HElib` version 2.0.0 [38]. Since we assume exact homomorphic encryption in this case, some parameters are different from those in the CKKS-FV framework. The LowMCv3 implementations were taken from the publicly available repositories (client-side implementation from `https://github.com/LowMC/lowmc` and server-side implementation from `https://bitbucket.org/malb/lowmc-helib`). The client-side implementations of FLIP were built from the publicly available repository (`https://github.com/CEA-LIST/Cingulata`).

6.1 Parameter Selection

Sets of parameters used in our implementation are given in Table 11, where

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

For the CKKS scheme, the message space is $\mathbb{C}^{\ell/2}$. When choosing parameters of CKKS-FV, one should be careful neither to overflow the plaintext modulus nor

to lose the required precision. In the following, we give a clear formula to choose t. Given N and p, we will fix positive integers c_l and c_u such that $c_u/c_l \leq c$ and for any message $\mathbf{m} = (m_1, \ldots, m_{N/2})$,

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

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

$$\frac{1}{c_l}\left(\frac{N}{\delta} + \frac{Nt}{2\Delta\delta}\right) < 2^{-p}$$

to preserve precision, and the plaintext modulus t such that $t > \sqrt{2}\delta c_u$ to bound the coefficients of the encoded plaintext. In summary, the plaintext modulus t should satisfy

$$t > 2^{p+1/2} c \left(N + \frac{Nt}{2\Delta}\right).$$

In Table 11, we recommend some parameters of HERA for $c = 4$ when combined with the CKKS-FV framework. In the case of encrypting short messages, we can choose an alternative set of parameters by changing the number of slots and the corresponding plaintext modulus t from Par-III. Unfortunately, there is no appropriate modulus t of 18 to 21 bits so we slightly change bits of precision. For $\lambda \in \{192, 256\}$, it suffices to use 6 and 7 rounds, respectively.

	λ	p	SKE		HE		
			$\lceil \log t \rceil$	r	$\log N$	$\log \ell$	$\lceil \log q \rceil$
Long Message							
Par-I	80	10	28	4	15	15	495
Par-II	80	14	32	4	15	15	550
Par-III	128	10	28	5	15	15	605
Par-IV	128	14	32	5	15	15	660
Short Message							
Par-A	128	9	17	5	15	4	495
Par-B	128	13	22	5	15	6	550
Par-C	128	11	22	5	15	8	550
Par-D	128	10	23	5	15	10	605

Table 11: Selected sets of parameters used in our implementation. 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.

6.2 Benchmarks

	Client-side		Server-side		Ctxt. Exp. Rate	
	Latency	Throughput	Latency	Throughput	HERA Ctxt.	CKKS Ctxt.
Long Message						
Par-I	4.687 ms	8.334 MB/s	32.62 s	9.772 KB/s	2.8	6.6
Par-II	4.964 ms	11.02 MB/s	37.27 s	11.81 KB/s	2.29	5.29
Par-III	4.801 ms	8.136 MB/s	51.49 s	6.660 KB/s	2.8	6.6
Par-IV	5.246 ms	10.42 MB/s	57.90 s	8.216 KB/s	2.29	5.29
Short Message						
Par-A	1.645 μs	10.44 MB/s	40.21 s	3.899 B/s	1.89	341.3
Par-B	7.654 μs	12.96 MB/s	45.94 s	19.47 B/s	1.69	142.8
Par-C	32.68 μs	10.27 MB/s	46.25 s	63.34 B/s	2	40.73
Par-D	136.6 μs	8.937 MB/s	52.24 s	207.5 B/s	2.3	11.6

Table 12: Performance of the CKKS-FV transciphering framework with HERA.

Performance of HERA with CKKS-FV Framework. As shown in Table 12, 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 pseudorandom numbers (needed to generate a single keystream in $\mathbb{Z}_t^N$), computation of E, FV-encoding, CKKS-encoding and vector addition over $\mathbb{Z}_t$. The extendable output function is instantiated with SHAKE256 in `XKCP`.

In Table 12, we also compare the ciphertext expansion rates of HERA and CKKS. The ciphertext size is 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 1024 no matter how short messages are encrypted; for short messages, the CKKS-ciphertext length cannot be proportional to the message length. We choose the ciphertext modulus of CKKS as small as possible while preserving the precision.

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-ciphertexts 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.

Cipher(Parameters)	Latency (cycles)	Throughput (C/B)
80-bit		
LowMCv3$(13, 128, 31)$	6.118×10^4	3824
LowMCv3$(13, 256, 49)$	1.757×10^5	5492
FLIP$(42, 128, {}^8\Delta^9)$	$\geq 4.303 \times 10^6$	$\geq 3.443 \times 10^7$
Rasta$(4, 327, 2)$	$\geq 2.224 \times 10^7$	$\geq 5.440 \times 10^5$
Rasta$(6, 219, 2)$	$\geq 1.320 \times 10^7$	$\geq 4.822 \times 10^5$
Dasta$(4, 327, 2)^\dagger$	8.648×10^4	2116
Dasta$(6, 219, 2)^\dagger$	7.246×10^4	2647
Masta$(4, 32, 65537)$	1.212×10^4	189.4
Masta$(5, 16, 65537)$	6004	187.6
HERA$(4, 16, 65537)$	4803	150.1
128-bit		
LowMCv3$(14, 196, 63)$	1.496×10^5	6106
LowMCv3$(14, 256, 63)$	1.947×10^5	6085
FLIP$(82, 224, {}^8\Delta^{16})$	$\geq 1.080 \times 10^7$	$\geq 8.643 \times 10^7$
Rasta$(5, 525, 2)$	$\geq 7.393 \times 10^7$	$\geq 1.127 \times 10^6$
Rasta$(6, 351, 2)$	$\geq 3.523 \times 10^7$	$\geq 8.029 \times 10^6$
Dasta$(5, 525, 2)^\dagger$	1.538×10^5	2344
Dasta$(6, 351, 2)^\dagger$	1.131×10^5	2579
Masta$(6, 32, 65537)$	1.373×10^4	214.5
Masta$(7, 16, 65537)$	6488	202.8
HERA$(5, 16, 65537)$	4911	153.5

†: These values are directly computed from the data in [37].

Table 13: Comparison of client-side performance between various HE-friendly ciphers.

Performance of HERA with BGV. We implemented HERA in the regular transciphering framework using the BGV scheme in `HElib`. Table 13 and 14 summarize the comparison of client-side and server-side performance between selected HE-friendly ciphers.

Cipher(Parameters)	$\lceil \log q \rceil$	N	Shape of Slots	λ'	Latency (s)	Throughput (KB/s)	Packing Method
80-bit							
LowMCv3$(13, 128, 31)$	380	18000	120×6	94.94	332.8	3.380×10^{-2}	row
LowMCv3$(13, 256, 49)$	410	18000	120×6	87.02	558.8	4.027×10^{-2}	row
FLIP$(42, 128, {}^8\Delta^9)$	60	4050	81	81.23	98.49	1.004×10^{-4}	row
FLIP$(42, 128, {}^8\Delta^9)$	110	15004	682	345.1	55.90	2.184×10^{-6}	col
FLIP$(42, 128, {}^8\Delta^9)$	60	4050	1	81.23	5.504	2.218×10^{-5}	no
Rasta$(4, 327, 2)$	170	7500	150	83.93	1822	3.286×10^{-3}	row
Rasta$(4, 327, 2)$	260	15004	682	115.9	265.7	3.005×10^{-4}	col
Rasta$(4, 327, 2)$	140	7500	1	89.98	72.83	5.481×10^{-4}	no
Rasta$(6, 219, 2)$	250	12000	60×10	88.63	3255	4.928×10^{-3}	row
Rasta$(6, 219, 2)$	320	15004	682	100.5	271.7	2.952×10^{-4}	col
Rasta$(6, 219, 2)$	180	12000	1	125.7	132.8	2.013×10^{-4}	no
Masta$(4, 32, 65537)$	400	16384	8192×2	115.5	261.3	3.919	row
Masta$(4, 32, 65537)$	420	16384	8192×2	80.23	36.87	0.8679	col
Masta$(5, 16, 65537)$	380	32768	16384×2	195.4	178.9	5.724	row
Masta$(5, 16, 65537)$	500	32768	16384×2	144.8	57.95	1.104	col
HERA$(4, 16, 65537)$	380	32768	16384×2	195.4	28.69	35.69	row
128-bit							
LowMCv3$(14, 196, 63)$	450	27000	$150 \times 6 \times 2$	132.2	1191	3.615×10^{-2}	row
LowMCv3$(14, 256, 63)$	450	27000	$150 \times 6 \times 2$	132.2	1337	4.206×10^{-2}	row
FLIP$(82, 224, {}^8\Delta^{16})$	110	7500	150	162.2	1195	1.532×10^{-5}	row
FLIP$(82, 224, {}^8\Delta^{16})$	120	49500	1650	714.9	736.1	1.658×10^{-7}	col
FLIP$(82, 224, {}^8\Delta^{16})$	110	7500	1	162.2	36.71	3.325×10^{-6}	no
Rasta$(5, 525, 2)$	230	14112	252×2	139.3	1.332×10^4	2.425×10^{-3}	row
Rasta$(5, 525, 2)$	320	43690	600×2	331.2	1517	8.449×10^{-5}	col
Rasta$(5, 525, 2)$	150	14112	1	187.1	310.9	2.061×10^{-4}	no
Rasta$(6, 351, 2)$	270	18000	120×6	140.1	1.198×10^4	2.576×10^{-3}	row
Rasta$(6, 351, 2)$	370	54000	360×6	334.2	1834	1.402×10^{-4}	col
Rasta$(6, 351, 2)$	190	18000	1	198.9	378.9	1.131×10^{-4}	no
Masta$(6, 32, 65537)$	450	32768	16384×2	169.5	768.7	2.664	row
Masta$(7, 16, 65537)$	500	32768	16384×2	144.8	241.0	4.249	row
HERA$(5, 16, 65537)$	490	32768	16384×2	147.9	36.64	27.95	row

Table 14: Comparison of server-side performance between various HE-friendly ciphers.

We note that Trivium and Kreyvium have not been included in the comparison since they are not suitable to generate a sufficiently long keystream from a single key. Dasta is included only in Table 13. The main purpose of Dasta is improvement of client-side performance, while it does not outperform Rasta on the server side. So we compare its client-side performance only using the data from the original paper [37].

In Table 13, we compare the client-side performance of HE-friendly ciphers. The benchmarks of LowMCv3 and FLIP have been computed using public repositories, while those of Rasta, Masta and HERA are obtained from our implementation using AVX2 instructions. For FLIP and Rasta, we only measured the time of "randomizing element" (permutation generator for FLIP, and XOF for Rasta) so that the benchmark will be better than practice, as marked by the inequality signs. For the parameters of the symmetric cipher, we remark the following.

- For LowMCv3(r, n, m), r, n and m denote the number of rounds, the block size, and the number of S-boxes in one round, respectively.
- For FLIP$(n_1, n_2, {}^{nb}\Delta^k)$, n_1, n_2, nb and k denote the number of bits in the linear part, the number of bits in the quadratic part, the number of triangular functions and their degree, respectively.
- For Rasta(r, n, t), Dasta(r, n, t), Masta(r, n, t) and HERA(r, n, t), r, n and t denote the number of rounds, the block size (in the number of words), and the plaintext modulus, respectively.

In Table 14, we compare the server-side performance of HE-friendly ciphers. The ciphertext modulus q is chosen with tight capacity, namely, homomorphic evaluation after transciphering is almost infeasible. In this table, 'Packing Method' stands for how we homomorphically evaluate ciphers (See Appendix A). In this column, 'row' (resp. 'col') means row-wise packing (resp. column-wise packing) and "no" represents no packing at all. The parameter λ' is the security level of the BGV scheme. The shape of slots represents a hypercube of a certain dimension which inherently supports the rotation along any axis. When adopting column-wise packing, this shape affects the throughput.

Packing methods affect the performance significantly. The difference mainly comes from the homomorphic evaluation of the linear layer, which is composed of multiplications by an $n \times n$ matrix. When multiplication by an $n \times n$ matrix is homomorphically evaluated as described in Appendix A (with size N), row-wise packing requires n^2 multiplications and $n(n-1)$ additions for N evaluations, while column-wise packing requires $2n-1$ rotations, $2n-1$ additions and $2n-1$ multiplications for N/n evaluations. With this observation, it will be reasonable to use column-wise packing for low latency, and row-wise packing for high throughput.

6.3 Considerations on Bootstrapping

In order to allow further computation over ciphertexts, sufficiently large parameters have been taken [2, 14, 25]. However, at the end of the CKKS-FV framework,

the final scaling factor $\delta \cdot \Delta$ and the last ciphertext modulus are so close to each other that any further arithmetic operation might be done modulo t. This means that this approach (of taking large parameters) is not applicable to the CKKS-FV framework, and one should bootstrap the ciphertexts at the end of the framework for further computation over ciphertexts.

Bootstrapping methods for the CKKS scheme have been actively studied [19, 16, 36], while all the methods use the sine function to approximate modulo q operation in the decryption circuit of the CKKS scheme. To make accurate approximation, it should be the case that $\|M\|_\infty \ll q$, where M is a (encoded) plaintext capsuled in the ciphertext to be bootstrapped and q is the ciphertext modulus after all the process of the CKKS-FV framework. Specifically, the sine function to approximate $[x]_q$ is given as

$$S(x) = \frac{q}{2\pi} \sin\left(\frac{2\pi x}{q}\right)$$

where the approximation error is bounded as follows.

$$|[x]_q - S(x)| = \frac{q}{2\pi}\left|\frac{2\pi x}{q} - \sin\left(\frac{2\pi x}{q}\right)\right| \leq \frac{2\pi^2}{3} \cdot \frac{|x|^3}{q^2}.$$

When we target $(p+1)$-bit precision, the plaintext M satisfies that $\|M\|_\infty \leq 2^{p+4}N$ assuming $c_u/c_l < 4$ and $\Delta \gg t$. As the plaintext is multiplied by Δ in the CKKS-FV framework, the approximation error is upper bounded by

$$\frac{2\pi^2}{3} \cdot \frac{|\Delta \cdot 2^{p+4}N|^3}{q^2}.$$

If this approximation error is upper bounded by $\Delta/2$, the decoded message will achieve p-bit precision. Letting $\Delta \approx q/t$, t should satisfy

$$t > \sqrt{\frac{2\pi^2}{3}} \cdot 2^{\frac{3p+12}{2}} N^{\frac{3}{2}}.$$

For example, in order to achieve 10-bit precision, t should be at least 45 bits.

In this work, we do not consider this issue in the choice of parameters. Instead, in order to make a fair comparison, we take "non-bootstrappable" parameters for every cipher when it is combined with CKKS. Recently, Bossuat et. al. [11] proposed a new bootstrapping technique for the RNS version of CKKS. We believe that application of this method to our framework will not significantly degrade the server-side throughput; based on their implementation result, we expect around 20% decrease.

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., 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 (2015)

[3] Aly, A., Ashur, T., Ben-Sasson, E., Dhooghe, S., Szepieniec, A.: Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols. IACR Transactions on Symmetric Cryptology **2020**(3) (Sep 2020)

[4] 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`

[5] 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)

[6] Bardet, M., Faugère, J.C., Salvy, B.: Asymptotic Behaviour of the Index of Regularity of Semi-Regular Quadratic Polynomial Systems. In: MEGA 2005 - 8th International Symposium on Effective Methods in Algebraic Geometry (2005)

[7] Bettale, L., Faugere, J.C., Perret, L.: Hybrid Approach for Solving Multivariate Systems over Finite Fields. Journal of Mathematical Cryptology **3**(3), 177–197 (2009)

[8] Bettale, L., Faugère, J.C., Perret, L.: Solving Polynomial Systems over Finite Fields: Improved Analysis of the Hybrid Approach. In: Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation. ISSAC '12, Association for Computing Machinery, New York, NY, USA (2012)

[9] Beyne, T., Canteaut, A., Dinur, I., Eichlseder, M., Leander, G., Leurent, G., Naya-Plasencia, M., Perrin, L., Sasaki, Y., Todo, Y., Wiemer, F.: Out of Oddity – New Cryptanalytic Techniques Against Symmetric Primitives Optimized for Integrity Proof Systems. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020. vol. 12172, pp. 299–328. Springer (2020)

[10] 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)

[11] Bossuat, J.P., Mouchet, C., Troncoso-Pastoriza, J., Hubaux, J.P.: Efficient Bootstrapping for Approximate Homomorphic Encryption with Non-Sparse Keys. Cryptology ePrint Archive, Report 2020/1203 (2020), `https://eprint.iacr.org/2020/1203`

[12] 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)

[13] 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)

[14] 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)

[15] Carlitz, L., Uchiyama, S.: Bounds for exponential sums. Duke mathematical Journal **24**(1), 37–41 (1957)

[16] 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)

[17] 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
[18] 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
[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.: TFHE: Fast Fully Homomorphic Encryption Over the Torus. Journal of Cryptology **33**(1), 34–91 (2020)
[24] 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)
[25] 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)
[26] 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)
[27] 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)
[28] Dworkin, M.J.: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Tech. rep., National Institute of Standards and Technology (2015)
[29] Fan, J., Vercauteren, F.: Somewhat Practical Fully Homomorphic Encryption. IACR Cryptology ePrint Archive, Report 2012/144 (2012), https://eprint.iacr.org/2012/144
[30] Fröberg, R.: An Inequality for Hilbert Series of Graded Algebras. MATHEMATICA SCANDINAVICA **56** (Dec 1985)
[31] 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)
[32] 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)

[33] 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)

[34] 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)

[35] Ha, J., Kim, S., Choi, W., Lee, J., Moon, D., Yoon, H., Cho, J.: Masta: An HE-Friendly Cipher Using Modular Arithmetic. IEEE Access **8**, 194741–194751 (2020)

[36] 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)

[37] Hebborn, P., Leander, G.: Dasta – Alternative Linear Layer for Rasta. IACR Transactions on Symmetric Cryptology **2020**(3), 46–86 (Sep 2020)

[38] HElib (release 2.0.0). `https://github.com/homenc/HElib` (Jan 2021)

[39] Hoffmann, C., Méaux, P., Ricosset, T.: Transciphering, Using FiLIP and TFHE for an Efficient Delegation of Computation. In: Bhargavan, K., Oswald, E., Prabhakaran, M. (eds.) Progress in Cryptology – INDOCRYPT 2020. pp. 39–61. Springer International Publishing, Cham (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] 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)

[45] 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)

[46] 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)

[47] Méaux, P., Carlet, C., Journault, A., Standaert, F.X.: Improved Filter Permutators for Efficient FHE: Better Instances and Implementations. In: Hao, F., Ruj, S., Sen Gupta, S. (eds.) Progress in Cryptology – INDOCRYPT 2019. vol. 11898, pp. 68–91. Springer (2019)

[48] 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)
[49] 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)
[50] 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)
[51] Rechberger, C., Soleimany, H., Tiessen, T.: Cryptanalysis of Low-Data Instances of Full LowMCv2. IACR Transactions on Symmetric Cryptology **2018**(3), 163–181 (2018)
[52] Regev, O.: On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. J. ACM **56**(6) (Sep 2009)
[53] Microsoft SEAL (release 3.4). `https://github.com/Microsoft/SEAL` (Oct 2019), microsoft Research, Redmond, WA.
[54] 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 $n \leq N$. 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

$$\begin{aligned}
&\mathsf{Ecd}^{\mathsf{FV}}\left(\mathsf{E}(\mathbf{m}^{(0)})_0, \ldots, \mathsf{E}(\mathbf{m}^{(N-1)})_0\right),\\
&\mathsf{Ecd}^{\mathsf{FV}}\left(\mathsf{E}(\mathbf{m}^{(0)})_1, \ldots, \mathsf{E}(\mathbf{m}^{(N-1)})_1\right),\\
&\qquad\vdots\\
&\mathsf{Ecd}^{\mathsf{FV}}\left(\mathsf{E}(\mathbf{m}^{(0)})_{n-1}, \ldots, \mathsf{E}(\mathbf{m}^{(N-1)})_{n-1}\right),
\end{aligned}$$

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

B On The Number of Monomials in 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 $\mathsf{MixRows} \circ \mathsf{MixColumns}$ has no zero entry. It implies that $\mathsf{MixRows} \circ \mathsf{MixColumns}$ 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

$$\begin{aligned}(a_0x_0 + a_1x_1 + \cdots + a_{n-1}x_{n-1} + b)^3 &= \sum_{i,j,k} a_ia_ja_kx_ix_jx_k + \sum_{i,j} 3a_ia_jbx_ix_j \\ &\quad + \sum_i 3a_ib^2x_i + b^3 \\ &= \sum_{i \leq j \leq k} \alpha(i,j,k)a_ia_ja_kx_ix_jx_k \\ &\quad + \sum_{i \leq j} \beta(i,j)a_ia_jbx_ix_j + \sum_i 3a_ib^2x_i + b^3,\end{aligned}$$

where

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

and

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

Since the plaintext modulus t is prime and $t > 2^{16}$, every monomial of degree 3 has a nonzero coefficient.

We can estimate the number of monomials in HERA with more rounds. Let $\mathbf{b} = (b_0, \ldots, b_{15})$ be the output of the first round function. The second round function will contain all the cubic monomials in $\mathbf{b}$. When we view the second round function as a polynomial in $b_0, \ldots, b_{15}$, some coefficients might become zero, while this happens only with probability of $1/t$. Heuristically (with the independence assumption), each monomial will remain at the second round with probability $1 - (1/t)^{16}$. This heuristic is confirmed by our computation, showing all possible monomials at the end of the second round. We conjecture that this property will hold for more than two rounds.

C 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

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

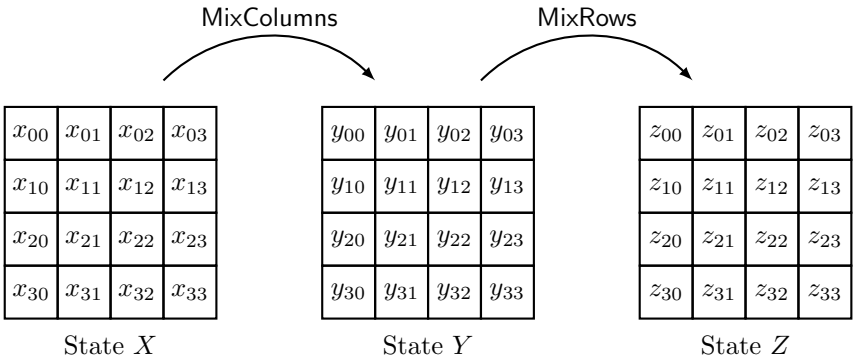


Fig. 7: Diagram of state change in HERA.

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 nonzero input and its output of $\mathbf{L}$. It is easily seen that $2 \leq B_\ell(\mathbf{M}), B_d(\mathbf{M}) \leq n+1$ for an invertible $n \times n$-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. Now we present the branch number of MixRows $\circ$ MixColumns.

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$\circ$MixColumns is 8. The linear branch number is computed similarly. We use the notations in Figure 7.

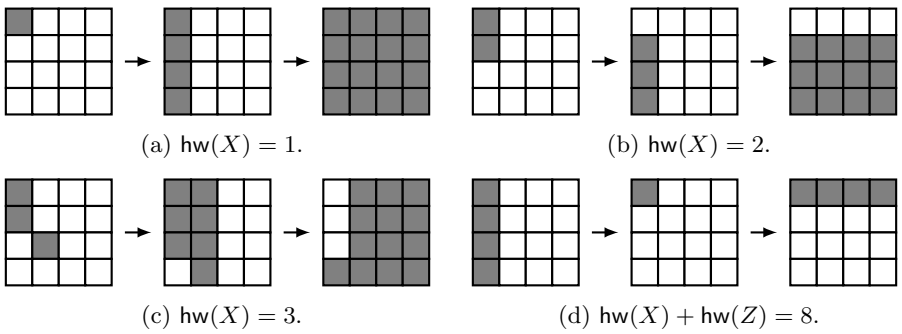
Suppose that the branch number B_d of MixRows $\circ$ MixColumns is less than 8. It means that there exists a triple of nonzero states (X, Y, Z) such that $\mathsf{hw}(X) + \mathsf{hw}(Z) \leq 7$. Assuming $\mathsf{hw}(X) \leq 3$, we distinguish the following three cases.

- $\mathsf{hw}(X) = 1$: all the components of Z are nonzero as seen in Figure 8a.

– $\mathsf{hw}(X) = 2$: two nonzero components might be in the same column or not. For either case, $\mathsf{hw}(Z) \geq 12$ (see Figure 8b 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 8c 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 8a, 8b, 8c) 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 8d. A specific example of Figure 8d 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. 8: Pictorial representation of four cases appearing in the proof of Theorem 2. A gray-colored cell represents a nonzero component.