Transciphering Framework for Approximate Homomorphic Encryption

Jihoon Cho[1], Jincheol Ha[2], Seongkwang Kim[2], Byeonghak Lee[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,lbh0307,hicalf}@kaist.ac.kr

Abstract. Homomorphic encryption (HE) is a promising cryptographic primitive that enables computation over encrypted data, with a variety of applications including medical, genomic, and financial tasks. In Asiacrypt 2017, Cheon et al. proposed the CKKS scheme to efficiently support approximate computation over encrypted data of real numbers. HE schemes including CKKS, nevertheless, still suffer from slow encryption speed and large ciphertext expansion compared to symmetric cryptography.

In this paper, we propose a novel hybrid framework, dubbed RtF (Real-to-Finite-field) framework, that supports CKKS. The main idea behind this construction is to combine the CKKS and the FV homomorphic encryption schemes, and use a stream cipher using modular arithmetic in between. As a result, real numbers can be encrypted without significant ciphertext expansion or computational overload on the client side.

As an instantiation of the stream cipher in our framework, we propose a new HE-friendly cipher, dubbed HERA, and extensively analyze its security and efficiency. The main feature of HERA is that it uses a simple randomized key schedule. Compared to recent HE-friendly ciphers such as FLIP and Rasta using randomized linear layers, HERA requires a smaller number of random bits. For this reason, HERA significantly outperforms existing HE-friendly ciphers on both the client and the server sides.

With the RtF transciphering framework combined with HERA at the 128-bit security level, we achieve small ciphertext expansion ratio with a range of 1.23 to 1.54, which is at least 23 times smaller than using (symmetric) CKKS-only, assuming the same precision bits and the same level of ciphertexts at the end of the framework. We also achieve 1.6 μs and 21.7 MB/s for latency and throughput on the client side, which are 9085 times and 17.8 times faster than the CKKS-only environment, respectively.

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

⋆ This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.2021R1F1A1047146).

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 while 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 [27], there has been a large amount of research in this area [10, 25, 18, 29]. Various applications of HE to medical, genomic, and financial tasks have also been proposed [15, 17, 37, 45].

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 [16] 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 [9, 46].

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 [28], 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 for Exact Computation. 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 [45] (see Figure 1). In the client-sever model, a client encrypts a message $\mathbf{m}$ using a symmetric cipher E with a secret key $\mathbf{k}$; this secret key is also encrypted using an HE

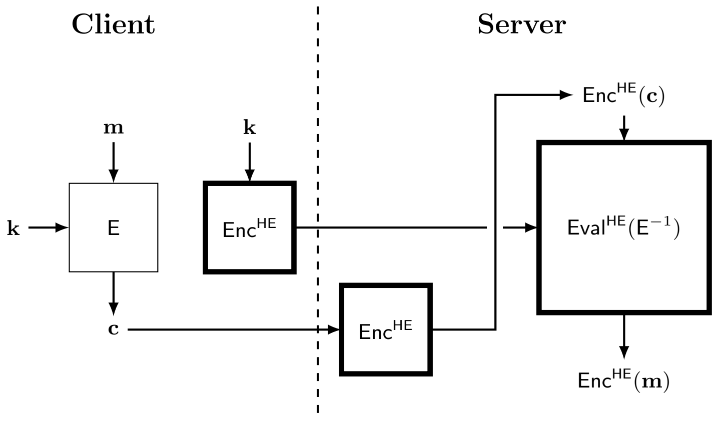
Client
Server
m
k
k
E
$\mathsf{Enc}^{\mathsf{HE}}$
c
$\mathsf{Enc}^{\mathsf{HE}}$
$\mathsf{Enc}^{\mathsf{HE}}(\mathbf{c})$
$\mathsf{Eval}^{\mathsf{HE}}(\mathsf{E}^{-1})$
$\mathsf{Enc}^{\mathsf{HE}}(\mathbf{m})$

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

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 [44] and Rasta [21].

1.1 Our Contribution

The main contribution of this paper is two-fold. First, we propose a new transciphering framework for the CKKS scheme that supports approximate computation over encrypted data. Second, we propose a new stream cipher, dubbed HERA (HE-friendly cipher with a RAndomized key schedule), to be built in our framework. Using our new transciphering framework combined with HERA, real numbers can be encrypted without significant ciphertext expansion or computational overload on the client side.

RtF Transciphering Framework. The transciphering framework in Figure 1 does not directly apply to the CKKS scheme. The main reason is that it is infeasible to design an HE-friendly (deterministic) symmetric cipher E operating on real (or complex) numbers; if an HE-friendly symmetric cipher E over the real field exists, then E is given as a real polynomial map, and any ciphertext will be represented by a polynomial in the corresponding plaintext and the secret key over $\mathbb{R}$. 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.

In order to overcome this problem, we combine CKKS with FV which is a homomorphic encryption scheme using modular arithmetic [25], obtaining a novel hybrid framework, dubbed the RtF (Real-to-Finite-field) transciphering framework. This framework inherits a wide range of usability from the previous transciphering framework, such as efficient short message encryption or flexible repacking of data on the server side. Additionally, our framework does not require to use the complex domain for message spaces (as in the CKKS scheme), or any expertise of the CKKS parameter setting on the client side.

In brief, the RtF framework works as follows. First, the client scales up and rounds off real messages into $\mathbb{Z}_t$. Then it encrypts the messages using a stream cipher E over $\mathbb{Z}_t$. This "E-ciphertext" will be sent to the server with an FV-encrypted secret key of E, and stored there.

Whenever a "CKKS-ciphertext" is needed for any message $\mathbf{m}$, the server encrypts the E-ciphertext of $\mathbf{m}$ in coefficients, using the FV scheme. With the resulting FV-ciphertext, say $\mathcal{C}$, and the FV-encrypted key, the server homomorphically evaluates the stream cipher E and moves the resulting keystreams from slots to coefficients using $\mathsf{SlotToCoeff}^{\mathsf{FV}}$. By subtracting this ciphertext from $\mathcal{C}$, the server obtains the FV-ciphertext of $\mathbf{m}$ in coefficients, not in slots. Finally, in order to translate this FV-ciphertext into the corresponding CKKS-ciphertext of $\mathbf{m}$ in slots, the server CKKS-bootstraps it. Since the message $\mathbf{m}$ should be moved from the coefficients to the slots, the last step of the bootstrapping, $\mathsf{SlotToCoeff}^{\mathsf{CKKS}}$, can be omitted. As a result, the server will be able to approximately evaluate any circuit on the CKKS-ciphertexts. Details of the framework are given in Section 3.

Low-depth Stream Ciphers Using Modular Arithmetic. In the RtF transciphering framework, a stream cipher using modular arithmetic is required as a building block. There are only a few ciphers using modular arithmetic [2, 4, 5, 30], 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, this type of ciphers spend too many random bits to generate 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. We implement the RtF transciphering framework with the stream cipher HERA in public repository[3]. In Section 5.2, we present the benchmark of the client-side encryption in C++ and the server-side transciphering using the `Lattigo` library. We also compare our framework to PEGASUS [40] and CKKS only. In the full version of this paper [19], we compare HERA to existing HE-friendly ciphers using the `HElib` library.

In summary, we achieve small ciphertext expansion ratio with a range of 1.23 to 1.54 on the client side, which is 23 times smaller than the (symmetric) CKKS-only environment assuming similar precision and the same level of ciphertexts at the end of the framework. When it comes to latency and throughput, we achieve 1.6 μs and 21.7 MB/s on the client side, which is 9085 times and 17.8 times faster than the CKKS-only environment respectively. We refer to Section 5.2 for more details.

1.2 Related Work

Homomorphic Evaluation of Symmetric Ciphers. Since the transciphering framework has been introduced [45], early works have been focused on homomorphic evaluation of popular symmetric ciphers (e.g., AES [28], SIMON [39], and PRINCE [23]). Such ciphers have been designed without any consideration on their arithmetic complexity, so the performance of their homomorphic evaluation

[3] `https://github.com/KAIST-CryptLab/RtF-Transciphering`

was not satisfactory. In this line of research, LowMC [3] 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 [20, 22, 47], so it has been revised later.[4]

Canteaut et al. [11] 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 [44] 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 [43], a generalized instantiation of FLIP, can be efficiently evaluated with the TFHE scheme [34]. Rasta [21] 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 [33], a variant of Rasta using affine layers with lower entropy, boosts up the client-side computation. As another variant of Rasta, Masta [31] operates on a modular domain, improving upon Rasta in terms of the throughput of homomorphic evaluation.

Compression of HE Ciphertexts. In order to reduce the memory overhead when encrypting short messages, Chen et al. [12] also proposed an efficient LWEs-to-RLWE conversion method which enables transciphering to the CKKS ciphertexts: small messages can be encrypted by LWE-based symmetric encryption with a smaller ciphertext size (compared to RLWE-based encryption), and a collection of LWE ciphertexts can be repacked to an RLWE ciphertext to perform a homomorphic evaluation. Lu et al. [40] proposed a faster LWEs-to-RLWE conversion algorithm in a hybrid construction of FHEW and CKKS, dubbed PEGASUS, where the conversion is not limited to a small number of slots.

Chen et al. [13] proposed a hybrid HE scheme using the CKKS encoding algorithm and a variant of FV. This hybrid scheme makes the ciphertext size a few times smaller compared to using CKKS only, in particular, when the number of slots is small. However, the ciphertexts from this hybrid scheme are of size larger than tens of kilobytes, which limits its practical relevance.

2 Preliminaries

Notations. 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 $\mathbb{Z} \cap (-q/2, q/2]$, and for any real number z, $[z]_q$ denotes the mod q reduction of z into $(-q/2, q/2]$. The notation $\lfloor \cdot \rceil$ and $[\cdot]_q$ are extended to vectors (resp. polynomials) to denote their component-wise (resp. coefficient-wise) reduction. For a complex vector $\mathbf{x}$, its ℓ_p-norm is denoted by $\|\mathbf{x}\|_p$. When

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

we say ℓ_p-norm of a polynomial, it means that the ℓ_p-norm of the coefficient vector of the polynomial.

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.

2.1 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 [48, 41]. For more details, we refer to [25, 16].

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 $\mathsf{Ecd}_\ell^{\mathsf{FV}} : \mathbb{Z}_t^\ell \to \mathcal{R}_t$ of the FV scheme is the inverse of the decoder $\mathsf{Dcd}_\ell^{\mathsf{FV}}$ defined by, for $p(X) = \sum_{k=0}^{\ell-1} a_k X^{kN/\ell} \in \mathcal{R}_t$,

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

where $\alpha_i = \xi^{5^i \cdot N/\ell} \pmod t$ for $0 \le i \le \ell/2 - 1$ and $\alpha_i = \xi^{-5^{i-\ell/2}\cdot N/\ell} \pmod t$ for $\ell/2 \le i \le \ell - 1$.[5]

Let Δ_{CKKS} be a positive real number (called a scaling factor in [16]). The CKKS encoder $\mathsf{Ecd}_{\ell/2}^{\mathsf{CKKS}} : \mathbb{C}^{\ell/2} \to \mathcal{R}$ is the (approximate) inverse of the decoder $\mathsf{Dcd}_{\ell/2}^{\mathsf{CKKS}} : \mathcal{R} \to \mathbb{C}^{\ell/2}$, where for $p(X) = \sum_{k=0}^{\ell-1} a_k X^{kN/\ell} \in \mathcal{R}$,

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

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

where $\beta_j = \zeta^{5^j \cdot N/\ell} \in \mathbb{C}$ for $0 \le j \le \ell/2 - 1$.

Algorithms. FV and CKKS share a common key generation algorithm. The descriptions of those two schemes 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 $0 \le i \le L-1$, and distributions $\mathcal{D}_{key}$, $\mathcal{D}_{err}$ and $\mathcal{D}_{enc}$ over $\mathcal{R}$ in a way that the resulting scheme is secure against any adversary with computational resource of $O(2^\lambda)$.
  1. Sample $a \leftarrow \mathcal{R}_{q_L}$, $s \leftarrow \mathcal{D}_{key}$, and $e \leftarrow \mathcal{D}_{err}$.
  2. The secret key is defined as $sk = (1, s) \in \mathcal{R}^2$, and the corresponding public key is defined as $pk = (b, a) \in \mathcal{R}_{q_L}^2$, where $b = [-a \cdot s + e]_{q_L}$.
  3. Sample $a' \leftarrow \mathcal{R}_{P \cdot q_L}$ and $e' \leftarrow \mathcal{D}_{err}$.
  4. The evaluation key is defined as $evk = (b', a') \in \mathcal{R}_{P \cdot q_L}^2$, where $b' = [-a' \cdot s + e' + Ps']_{P \cdot q_L}$ for $s' = [s^2]_{q_L}$.
- Encryption: given a public key $pk \in \mathcal{R}_{q_L}^2$ 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, $\mathsf{Enc}^{\mathsf{FV}}(pk, m) = [\mathsf{Enc}(pk, 0) + (\Delta_{\mathsf{FV}} \cdot [m]_t, 0)]_{q_L}$, where $\Delta_{\mathsf{FV}} = \lfloor q_L/t \rceil$.
  - 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 $ct \in \mathcal{R}_{q_l}^2$,

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

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

$$ct_{add} = [ct_1 + ct_2]_{q_l}.$$

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

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

  where (d_0, d_1, d_2) is defined by $[(b_1 b_2, a_1 b_2 + a_2 b_1, a_1 a_2)]_{q_l}$ when using CKKS and $\left[\left\lfloor \frac{t}{q_l} (b_1 b_2, a_1 b_2 + a_2 b_1, a_1 a_2) \right\rceil \right]_{q_l}$ when using FV.
- Rescaling (Modulus switching): given a ciphertext $ct \in \mathcal{R}_{q_l}^2$ and $l' < l$, its rescaled ciphertext is defined as

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

2.2 Some Notable Homomorphic Operations

Bootstrapping for CKKS. The bootstrapping procedure for CKKS has been actively studied recently [32, 8, 38, 14]. Let ct be a CKKS-ciphertext of $m(Y) \in \mathbb{Z}[Y]/(Y^\ell + 1)$ with respect to the secret key sk and the ciphertext modulus q, where $Y = X^{N/\ell}$, namely, $m(Y) = [\langle ct, sk\rangle]_q$. In this case, $m(Y)$ has $\ell/2$ slots. The CKKS bootstrapping aims to find a larger modulus $Q > q$ and a ciphertext ct' such that $m(Y) = [\langle ct', sk\rangle]_Q$. It consists of five steps: $\mathsf{ModRaise}$, SubSum, $\mathsf{CoeffToSlot}^{\mathsf{CKKS}}$, $\mathsf{EvalMod}$, and $\mathsf{SlotToCoeff}^{\mathsf{CKKS}}$.

- $\mathsf{ModRaise}$: If we set $t(X) = \langle ct, sk\rangle \in \mathcal{R}$, then $t(X) = q \cdot I(X) + m(Y)$ for some $I(X) \in \mathcal{R}$. $\mathsf{ModRaise}$ raises the ciphertext modulus to $Q \gg q$ so that ct is regarded as an encryption of $t(X)$ with respect to modulus Q.
- SubSum: If $N \neq \ell$, then SubSum maps $I(X)$ to a polynomial in Y, that is, $q \cdot I(X) + m(Y)$ to $(N/\ell) \cdot (q \cdot \tilde{I}(Y) + m(Y))$.
- $\mathsf{CoeffToSlot}^{\mathsf{CKKS}}$: Since the message $q \cdot I(X) + m(Y)$ is in the coefficient domain, it requires homomorphic evaluation of the encoding algorithm to enable slot-wise modulo q operation. $\mathsf{CoeffToSlot}^{\mathsf{CKKS}}$ performs homomorphic evaluation of the inverse Discrete Fourier Transform (DFT) to obtain the ciphertext(s) of $\mathsf{Ecd}^{\mathsf{CKKS}}(q \cdot I(X) + m(Y))$.
- $\mathsf{EvalMod}$: To approximate the modulo q operation, $\mathsf{EvalMod}$ homomorphically evaluates a polynomial approximation of $f(t) = \frac{q}{2\pi} \sin\left(\frac{2\pi t}{q}\right)$. In recent works [8, 32], Chebyshev polynomial approximations are used.
- $\mathsf{SlotToCoeff}^{\mathsf{CKKS}}$: It performs homomorphic evaluation of DFT to output a ciphertext of $m(Y)$ back in its coefficient domain.

Operations in FV. In the FV scheme, there are two operations between slots and coefficients.

- $\mathsf{CoeffToSlot}^{\mathsf{FV}}$: It is a homomorphic evaluation of FV-encoding function. It semantically puts the coefficients of a plaintext polynomial into the vector of slots. It is done by multiplying the inverse Number Theoretic Transform (NTT) matrix.
- $\mathsf{SlotToCoeff}^{\mathsf{FV}}$: It is a homomorphic evaluation of FV-decoding function. It semantically puts the slot vector of a message into the coefficients of the plaintext polynomial. It is also done by multiplying the NTT matrix.

3 RtF Transciphering Framework

In this section, we describe how the RtF transciphering framework works, and analyze the message precision of the framework.

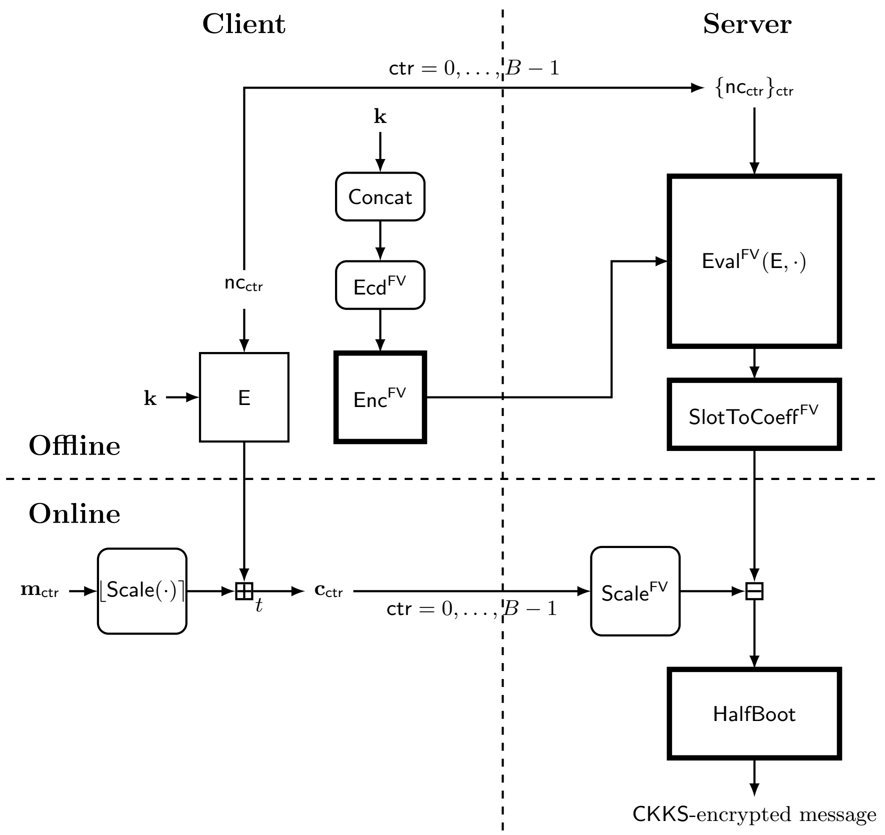
Client
Server
$\mathsf{ctr} = 0, \ldots, B-1$
$\{\mathsf{nc}_{\mathsf{ctr}}\}_{\mathsf{ctr}}$
$\mathbf{k}$
Concat
$\mathsf{Ecd}^{\mathsf{FV}}$
$\mathsf{nc}_{\mathsf{ctr}}$
$\mathsf{Eval}^{\mathsf{FV}}(\mathsf{E}, \cdot)$
$\mathbf{k}$
E
$\mathsf{Enc}^{\mathsf{FV}}$
$\mathsf{SlotToCoeff}^{\mathsf{FV}}$
Offline
Online
$\mathbf{m}_{\mathsf{ctr}}$
$\lfloor \mathsf{Scale}(\cdot) \rceil$
t
$\mathbf{c}_{\mathsf{ctr}}$
$\mathsf{ctr} = 0, \ldots, B-1$
$\mathsf{Scale}^{\mathsf{FV}}$
HalfBoot
CKKS-encrypted message

Fig. 2: The RtF 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. The client sends ciphertexts block by block, while the server gathers B ciphertext blocks and recovers the CKKS-encrypted message of the ciphertexts.

3.1 Overview of the Framework

Our RtF transciphering framework aims to replace the (basic) transciphering framework in Figure 1 to support CKKS, when equipped with any suitable stream cipher. The overall design is depicted in Figure 2. At a high level, we propose to use a stream cipher operating on $\mathbb{Z}_t^n$ to encrypt real number messages on the client side and to convert the ciphertexts into the corresponding CKKS ciphertexts on the server side. In this regard, it is required to employ an additional HE scheme which provides homomorphic evaluation of keystreams of the stream cipher over the modulo t spaces efficiently, and we use FV for this purpose.

The main idea of the RtF framework is to inject real messages into the coefficients of plaintext polynomials of FV and to delegate encoding/decoding to the server via $\mathsf{SlotToCoeff}$ and $\mathsf{CoeffToSlot}$ for FV and CKKS which is described more precisely as follows.

First, a message of real numbers $\mathbf{m}_{\mathsf{ctr}} \in \mathbb{R}^n$ is scaled into $\mathbb{Z}_t^n$ by multiplying by a constant and rounding, and encrypted to $\mathbf{c}_{\mathsf{ctr}}$ on the client side. After gathering symmetric ciphertexts $\mathbf{c}_{\mathsf{ctr}}$'s from the client, the server generates a polynomial $C \in \mathcal{R}_t$ whose coefficients are components of $\mathbf{c}_{\mathsf{ctr}}$'s. Then the polynomial is scaled up into the FV ciphertext space by multiplying Δ_{FV}, say $\mathcal{C} = (\Delta_{\mathsf{FV}} \cdot C, 0)$.[6] On the other hand, when the server evaluates the symmetric cipher, a bunch of the keystream is FV-encrypted in slots. In order to match the domain of computation, the server evaluates

$$\mathsf{SlotToCoeff}^{\mathsf{FV}} : \mathsf{Enc}^{\mathsf{FV}}(\mathsf{Ecd}^{\mathsf{FV}}(z_0, \ldots, z_{N-1})) \mapsto \mathsf{Enc}^{\mathsf{FV}}(z_0 + \cdots + z_{N-1}X^{N-1})$$

after evaluation of the cipher, where $(z_0, \ldots, z_{N-1})$ is the concatenated keystream. Then, homomorphically computing

$$(\Delta_{\mathsf{FV}} \cdot C, 0) - \mathsf{Enc}^{\mathsf{FV}}(z_0 + \cdots + z_{N-1}X^{N-1}),$$

we have $\mathsf{Enc}^{\mathsf{FV}}(m_0 + \cdots + m_{N-1}X^{N-1})$, where $(m_0, \ldots, m_{N-1})$ is the concatenated message. The next step is to convert the type of encryption to CKKS and then to put the messages into slots, which can be done by $\mathsf{HalfBoot}$.

In the bootstrapping procedure, there are five steps as follows:

$$\mathsf{ModRaise} \to \mathsf{SubSum} \to \mathsf{CoeffToSlot}^{\mathsf{CKKS}} \to \mathsf{EvalMod} \to \mathsf{SlotToCoeff}^{\mathsf{CKKS}}.$$

$\mathsf{HalfBoot}$ basically follows the procedure of CKKS bootstrapping, except the final $\mathsf{SlotToCoeff}^{\mathsf{CKKS}}$ step. Since the input ciphertext of $\mathsf{HalfBoot}$ contains the original message $(m_0, \ldots, m_{N-1})$ in coefficients rather than slots, it does not require to move data in slots back to coefficients after $\mathsf{EvalMod}$. Furthermore, with an appropriate rescaling, $\mathsf{HalfBoot}$ gives an effect of full bootstrapping to enable further approximate computations on the output CKKS ciphertexts.

[6] We note that $\mathbf{c}_{\mathsf{ctr}}$'s are in coefficients, not in slots.

3.2 Specification

For a fixed security parameter λ, all the other parameters for the FV and the 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 parameters fixed, we will describe how the framework works, distinguishing four parts; initialization, client-side computation, and offline/online server-side computation (see Figure 2). The client-side and server-side computations are described in Algorithm 1 and Algorithm 2, respectively.

Algorithm 1: Client-side symmetric key encryption of the RtF transciphering framework

```
Input:
  – Nonce $\mathsf{nc}_{\mathsf{ctr}} \in \{0,1\}^{\lambda}$
  – Symmetric key $\mathbf{k} \in \mathbb{Z}_t^n$
  – Tuple of messages $\mathbf{m}_{\mathsf{ctr}} \in \mathbb{R}^n$
  – Scaling factor $\delta$
Output:
  – Symmetric ciphertext $\mathbf{c}_{\mathsf{ctr}} \in \mathbb{Z}_t^n$

1 $\mathbf{z}_{\mathsf{ctr}} \leftarrow \mathsf{E}(\mathbf{k}_{\mathsf{ctr}}, \mathsf{nc}_{\mathsf{ctr}})$
2 $\widetilde{\mathbf{m}}_{\mathsf{ctr}} \leftarrow \lfloor \delta \cdot \mathbf{m}_{\mathsf{ctr}} \rceil$
3 $\mathbf{c}_{\mathsf{ctr}} \leftarrow [\widetilde{\mathbf{m}}_{\mathsf{ctr}} + \mathbf{z}_{\mathsf{ctr}}]_t$
4 return $\mathbf{c}_{\mathsf{ctr}}$
```

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. Let ℓ be the number of used slots per FV-ciphertext to encrypt $\mathbf{k} \in \mathbb{Z}_t^n$ which satisfies $n \mid \ell$ and $\ell \mid N$. To enable SIMD evaluation for keystreams, we consider the following matrix of B duplications of $\mathbf{k}$.

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

The client can pack the coefficients of matrix $\mathsf{Concat}(\mathbf{k})$ column-wisely into one glued column vector in $\mathbb{Z}_t^{nB}$ or row-by-row manner, which are called column-wise and row-wise packing, respectively. The number of keystreams calculated in a single ciphertext (resp. n ciphertexts) is $B = \ell/n$ for column-wise packing (resp. $B = \ell$ for row-wise packing). We refer to the full version [19] for more details.

To summarize, the client computes

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

Algorithm 2: Server-side homomorphic evaluation of decryption of the RtF transciphering framework

Input:

- Set of nonces $\mathsf{nc}_0, \ldots, \mathsf{nc}_{B-1} \in \{0,1\}^\lambda$
- Homomorphically encrypted keys $\mathcal{K} = \mathsf{Enc}^{\mathsf{FV}}\left(\mathsf{Ecd}^{\mathsf{FV}}(\mathsf{Concat}(\mathbf{k}))\right)$
- Tuple of symmetric ciphertexts $\mathbf{c} = (\mathbf{c}_0, \ldots, \mathbf{c}_{B-1}) \in (\mathbb{Z}_t^n)^B$

Output:

- CKKS-encrypted message $\mathcal{M}$

```
1 $\mathcal{V} \leftarrow \mathsf{Eval}^{\mathsf{FV}}(\mathsf{E}, \mathcal{K}, \{\mathsf{nc}_{\mathsf{ctr}}\}_{\mathsf{ctr}})$
2 $\mathcal{Z} \leftarrow \mathsf{SlotToCoeff}^{\mathsf{FV}}(\mathcal{V})$
3 $C \leftarrow \mathsf{VecToPoly}(\mathbf{c})$
4 $\mathcal{C} \leftarrow (\Delta_{\mathsf{FV}} \cdot C, 0)$
5 $\mathcal{X} \leftarrow [\mathcal{C} - \mathcal{Z}]_q$
6 $\mathcal{X} \leftarrow \mathsf{Rescale}_{\to 0}(\mathcal{X})$          // Rescale to the lowest level
7 $\mathcal{M} \leftarrow \mathsf{HalfBoot}(\mathcal{X})$
8 return $\mathcal{M}$
```

and sends $\mathcal{K}$ to the server. We note that this initialization phase can be done only once at the beginning of the RtF 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-tuple of real messages $\mathbf{m} = (m_0, \ldots, m_{n-1}) \in \mathbb{R}^n$, and a scaling factor $\delta > 0$, the client executes the following encryption algorithm as described in Algorithm 1.

The client computes keystream $\mathbf{z} = \mathsf{E}_{\mathbf{k}}(\mathsf{nc}) \in \mathbb{Z}_t^n$. Then, the client scales the message $\mathbf{m}$ by multiplying δ to every component of $\mathbf{m}$. Rounding it off gives a vector $\widetilde{\mathbf{m}} \in \mathbb{Z}^n$. If t and δ are appropriately chosen, the norm $\|\widetilde{\mathbf{m}}\|_\infty$ can be upper bounded by $t/2$. Finally, the client computes

$$\mathbf{c} \coloneqq [\widetilde{\mathbf{m}} + \mathbf{z}]_t\,,$$

and sends it to the server.

Offline Server-side Computation. Given a tuple of nonces $(\mathsf{nc}_0, \ldots, \mathsf{nc}_{B-1})$ and the FV-encrypted key $\mathcal{K}$, the server is able to construct a circuit for the homomorphic evaluation of E, denoted by $\mathsf{Eval}^{\mathsf{FV}}(\mathsf{E}, \{\mathsf{nc}_{\mathsf{ctr}}\}_{\mathsf{ctr}}, \cdot)$. The circuit constructed for column-wise (resp. row-wise) packing method returns 1 ciphertext (resp. n ciphertexts) which packs ℓ/n keystreams (resp. ℓ keystreams). With the FV-encrypted key $\mathcal{K}$, the server homomorphically computes $\mathcal{V} \coloneqq \mathsf{Eval}^{\mathsf{FV}}(\mathsf{E}, \{\mathsf{nc}_{\mathsf{ctr}}\}_{\mathsf{ctr}}, \mathcal{K})$. For ease of notation, we explain the remaining parts with column-wise packing method. Denoting the concatenation of ℓ/n keystreams by

$(z_0, \ldots, z_{\ell-1}) \in \mathbb{Z}_t^\ell$, the resulting FV-ciphertext $\mathcal{V}$ can be represented as

$$\mathsf{Enc}^{\mathsf{FV}}\left(\mathsf{Ecd}_\ell^{\mathsf{FV}}(z_0, \ldots, z_{\ell-1})\right).$$

Finally, the server computes

$$\mathcal{Z} \coloneqq \mathsf{SlotToCoeff}^{\mathsf{FV}}(\mathcal{V}) = \mathsf{Enc}^{\mathsf{FV}}\left(\sum_{k=0}^{\ell-1} z_k X^{k \cdot N/\ell}\right).$$

Online Server-side Computation. Given a tuple of symmetric ciphertexts $\mathbf{c} = (\mathbf{c}_0, \ldots, \mathbf{c}_{\ell/n-1}) \in (\mathbb{Z}_t^n)^{\ell/n}$, the server scales up $\mathbf{c}$ into FV-ciphertext space to enable FV evaluation, namely

$$C \coloneqq \mathsf{VecToPoly}(\mathbf{c}),$$
$$\mathcal{C} \coloneqq (\Delta_{\mathsf{FV}} \cdot C, 0),$$

where $\mathsf{VecToPoly}$ is defined by

$$\begin{aligned} \mathsf{VecToPoly}: \qquad \mathbb{R}^\ell &\longrightarrow \mathbb{R}[X]/(\Phi_{2N}(X)) \\ (m_0, \ldots, m_{\ell-1}) &\mapsto \sum_{k=0}^{\ell-1} m_k X^{k \cdot N/\ell}. \end{aligned}$$

Then, server computes $\mathcal{X} \coloneqq [\mathcal{C} - \mathcal{Z}]_q$, where q is the ciphertext modulus of $\mathcal{Z}$, and rescales it to the lowest level of CKKS.

Now, the only remaining procedure is $\mathsf{HalfBoot}$, which combines $\mathsf{ModRaise}$, SubSum, $\mathsf{CoeffToSlot}^{\mathsf{CKKS}}$, and $\mathsf{EvalMod}$ sequentially. Denoting the scaled message by $(\widetilde{\mathbf{m}}_0, \ldots, \widetilde{\mathbf{m}}_{\ell/n-1}) \coloneqq (\widetilde{m}_0, \ldots, \widetilde{m}_{\ell-1}) \in \mathbb{Z}^\ell$, the resulting ciphertext can be represented as

$$\mathcal{X} \coloneqq \mathsf{Enc}^{\mathsf{CKKS}}\left(\sum_{k=0}^{\ell-1} \widetilde{m}_k X^{k \cdot N/\ell}\right).$$

Then, after $\mathsf{ModRaise}$, we have

$$\mathcal{X}' \coloneqq \mathsf{Enc}^{\mathsf{CKKS}}\left(\sum_{k=0}^{\ell-1} \widetilde{m}_k X^{k \cdot N/\ell} + q_0 \cdot I(X)\right)$$

for some polynomial $I(X) = r_0 + \cdots + r_{N-1}X^{N-1} \in \mathcal{R}$. By evaluating SubSum, the polynomial $I(X)$ becomes sparsely packed

$$\widetilde{I}(X) = \frac{N}{\ell} \sum_{k=0}^{\ell-1} r_{k \cdot N/\ell} X^{k \cdot N/\ell}$$

and the message is scaled by N/ℓ, say $\widetilde{m}_k \leftarrow (N/\ell) \cdot \widetilde{m}_k$. Evaluating $\mathsf{CoeffToSlot}^{\mathsf{CKKS}}$ gives two ciphertexts as follows.

$$\mathcal{Y}_0 = \mathsf{Enc}^{\mathsf{CKKS}}\left(\mathsf{Ecd}_{\ell/2}^{\mathsf{CKKS}}(\widetilde{m}_0 + q_0\widetilde{r}_0, \ldots, \widetilde{m}_{\ell/2-1} + q_0\widetilde{r}_{\ell/2-1})\right)$$
$$\mathcal{Y}_1 = \mathsf{Enc}^{\mathsf{CKKS}}\left(\mathsf{Ecd}_{\ell/2}^{\mathsf{CKKS}}(\widetilde{m}_{\ell/2} + q_0\widetilde{r}_{\ell/2}, \ldots, \widetilde{m}_{\ell-1} + q_0\widetilde{r}_{\ell-1})\right)$$

where $\widetilde{r}_k = (N/\ell) \cdot r_{k \cdot N/\ell}$ for $k = 0, 1, \ldots, \ell - 1$. If $\ell \neq N$, then those two ciphertexts can be packed in a ciphertext. As $\mathsf{EvalMod}$ evaluates the modulo-q_0 operation approximately, $\mathsf{EvalMod}$ operation results in what we want.

3.3 Message Precision

As the CKKS scheme adds some noise for every arithmetic operation, it is important to analyze how close the output $\mathcal{M}$ of Algorithm 2 is to the original message. In this section, we bound the error occurred in the RtF framework. First, we bound the error in the middle state $\mathcal{X}$ in Algorithm 2.

Let $\mathbf{m} \in \mathbb{R}^\ell$ be an (ℓ/n)-concatenation of the client's message as an input to Algorithm 1 such that $\widetilde{\mathbf{m}} = \lfloor \delta \cdot \mathbf{m} \rceil$ and $\|\widetilde{\mathbf{m}}\|_\infty \leq \lfloor t/2 \rfloor$, and let $\mathcal{X}$ be the state in line 5 of Algorithm 2 before rescaling to zero level and $\mathsf{HalfBoot}$ in Algorithm 2. If $e_{\mathsf{eval}} \in \mathcal{R}$ is an error from homomorphic evaluation of E with FV such that $\|e_{\mathsf{eval}}\|_\infty < \Delta_{\mathsf{FV}}/2$ (i.e., the ciphertext is correctly FV-decryptable), then we have

$$\begin{aligned}\left\| \mathsf{VecToPoly}(\mathbf{m}) - \frac{\mathsf{Dec}^{\mathsf{CKKS}}(\mathcal{X})}{\delta \Delta_{\mathsf{FV}}} \right\|_\infty &\leq \frac{1}{2\delta} + \frac{\|e_{\mathsf{eval}}\|_\infty}{\Delta_{\mathsf{FV}} \delta} \\ &\leq \frac{1}{2\delta} + \frac{1}{2\delta} = \frac{1}{\delta}\end{aligned}$$

since $\|\mathbf{m} - \widetilde{\mathbf{m}}/\delta\|_\infty \leq \frac{1}{2\delta}$ and $[\widetilde{\mathbf{m}}]_t = \widetilde{\mathbf{m}}$. We remark that e_{eval} depends on the construction of E, which will be bounded appropriately for our new stream cipher and proposed parameters for HE.

The change of the message precision in $\mathsf{HalfBoot}$ varies according to which specific algorithm is used. We basically follow the work of Bossuat et al. [8] of CKKS bootstrapping, and we describe the message precision using those results.

In the bootstrapping procedure, the most significant step is to approximate modular reduction, which corresponds to $\mathsf{EvalMod}$. As modular reduction itself is not well-matched with polynomial approximation, the sine function is commonly used as a stepping-stone to evaluate modular reduction in bootstrapping algorithms. As a result, there are two kinds of error to be considered in $\mathsf{EvalMod}$: one from distance between modular reduction and sine function, and the other from polynomial approximation of the sine function.

The first one, from distance between modular reduction and the sine function, is mainly determined by the ratio of the bootstrapping scaling factor Δ' to the modulus q_0. Bootstrapping algorithms use scaling factor Δ' larger than default scaling factor Δ_{CKKS} used for basic arithmetic, since approximating modular reduction induces much larger error. In this case, the distance between the modular reduction and the sine function is bounded by Taylor's theorem as follows.

$$\left| \frac{q_0}{\Delta'} \left[\frac{\Delta'}{q_0} x \right]_1 - \frac{q_0}{2\pi\Delta'} \sin\left(\frac{2\pi\Delta' x}{q_0} \right) \right| \leq \frac{q_0}{2\pi\Delta'} \cdot \frac{1}{3!} \left(\frac{2\pi\Delta'}{q_0} \right)^3 = \frac{2\pi^2}{3} \left(\frac{\Delta'}{q_0} \right)^2 \quad (1)$$

provided that $|x| \leq 1$.

The other error from polynomial approximation of the sine function is determined by the polynomial interpolation algorithms. In this step, Bossuat et al. [8] adopt a specialized Chebyshev interpolation proposed by Han and Ki [32] for sparse keys, and combine it with their optimization method, which is called errorless polynomial evaluation. The error bound is calculated based on the distribution of Chebyshev nodes which is empirically achieved, and we recommend to see [32] for further discussion. Similarly to (1), this error bound also decreases when Δ'/q_0 gets smaller. Thus, we present an experimental result of correlation between Δ'/q_0 and the message precision in Table 1.

Δ'/q_0	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}
$-\log\varepsilon$	11.29	13.29	15.30	17.29	19.28	21.24	22.73

Table 1: This table presents experimental error of HalfBoot for various Δ'/q_0. The value ε is the mean error occurred by HalfBoot. The experiment is done by using parameter `Par-128` in Table 4 except Δ'/q_0.

In our transciphering framework, the value Δ'/q_0 is approximately δ/t. The plaintext modulus t should be larger than the number of precision bits, which is the reason for ciphertext expansion in our framework. This expansion can be reduced when arcsin is evaluated after the sine function.

After HalfBoot, we obtain a refreshed CKKS ciphertext of pre-determined scale Δ_{CKKS} as a result of RtF framework. Although we can freely choose the final scale Δ_{CKKS}, the message precision of the RtF framework cannot exceed $\log\delta$ bits. Hence it is enough to choose $\Delta_{\mathsf{CKKS}} \gg \delta N$ to ensure maximum precision $\log\delta$ against scaling error.

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

The RtF 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

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

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

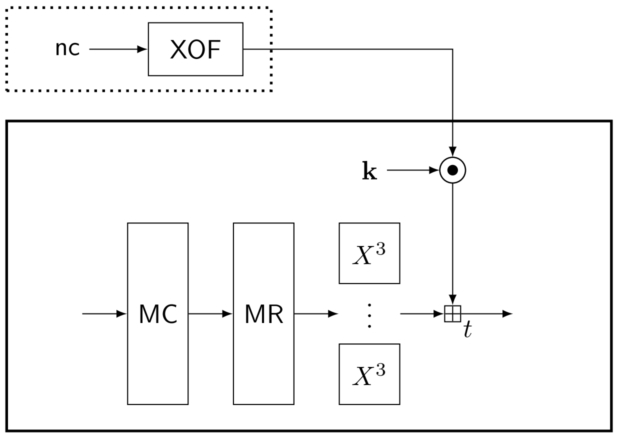

Fig. 3: The round function of HERA. Operations in the box with dotted (resp. thick) lines are public (resp. secret). "MC" and "MR" represent MixColumns and MixRows, respectively.

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.

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

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 [24].

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 $\mathbb{Z}_t$.

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

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

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

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

(a) MixColumns

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

(b) MixRows

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

$\mathbf{x} = (x_0, \ldots, x_{15}) \in \mathbb{Z}_t^{16}$, we have

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

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 [44] and Rasta [21]). 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 [44] or Rasta [21] 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 2, we compare different types of linear maps to the (nonlinear) Cube map in terms of evaluation time and noise budget 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. 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 (uniformly sampled) fixed linear layer due to its sparsity.

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

Table 2: Comparisons of different types of maps in terms of evaluation time and noise budget 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 the full version [19]) 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 [42], recently attracting renewed interest for the use in MPC/ZKP-friendly ciphers [2, 4]. The cube map has good linear/differential characteristics, whose inverse is of high degree, mitigating meet-in-the-middle algebraic attacks.

As multiplicative depth heavily impacts on noise growth of HE-ciphertexts, it is desirable to design HE-friendly ciphers using a small number of rounds. One of the most threatening attacks on ciphers with low algebraic degrees is the higher order differential attack. For a λ-bit secure (possibly non-binary) cipher, the algebraic degree of the cipher should be at least $\lambda - 1$. However, the attack is not available on randomized 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 [36]. 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 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 the full version [19]). Based on this observation, we opt for component-wise multiplication. It offers better security on algebraic attacks and linear cryptanalysis. For a traditional block cipher using fixed keys, outer affine layers do not affect its overall security; when it comes to HERA, the first and the last affine layers, combined with the randomized key schedule, increases the number of monomials.

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.

4.3 Security Analysis of HERA

In this section, we provide a summary of the security analysis of HERA (due to the page limit). All the details are given in the full version [19]. Table 3 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 $t > 2^{16}$.

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

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 3: Recommended number of rounds with respect to each attack.

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 [7] 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.

Trivial Linearization. Trivial linearization is to solve a system of linear equations by replacing all monomials by new variables. When applied to the r-round HERA cipher, the number of monomials appearing in this system is upper bounded by

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

Therefore, at most S equations will be enough to solve this system of equations. All the monomials of degree at most 3^r are expected to appear after r rounds of HERA (as explained in detail in the full version [19]). 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. However, this strategy will not be useful when $t > 2^{16}$.

GCD Attack. The GCD attack seeks to compute the greatest common divisor (GCD) of univariate polynomials, and it can be useful for a cipher operating on a large field with its representation being a polynomial in a single variable. 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 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.

Gröbner Basis Attack. The Gröbner basis attack is to solve a system of equations by computing a Gröbner basis of the system. We analyze the security of HERA against the Gröbner basis attack under the semi-regular assumption, which is reasonable as conjectured in [26].

The degree of regularity of the system can be computed as the degree of the first non-positive coefficient in the Hilbert series

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

where r is the number of rounds and m is the number of equations. 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. Therefore, the time complexity of the Gröbner basis attack is lower bounded by

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

Any variant based on the guess-and-determine strategy requires even higher complexity when $r \leq 6$. 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. In this case, the complexity is estimated as

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

In the full version [19], we compute the degree $d_{reg}(r,k)$ of regularity and estimate the time complexity of the attack.

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 [35]. It is known that the data complexity of this attack depends on the number of monomials in the polynomial representation of the 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

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

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 the full key space.

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.

Linear Cryptanalysis. Linear cryptanalysis typically applies to block ciphers operating on binary spaces. However, linear cryptanalysis can be extended to non-binary spaces [6]; 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 be 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 denoted by

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

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 [6]. 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 $1/(\mathrm{LP}^S)^{B_\ell\cdot\lfloor\frac{r}{2}\rfloor}$. Again, we have $\mathrm{LP}^S\leq 4/t$ as seen in the full version [19]. As the branch number of the linear layer of HERA is 8 (as shown in the full version [19]), we can conclude that r-round HERA provides λ-bit security against linear cryptanalysis when

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

5 Implementation

In this section, we evaluate the performance of the RtF framework combined with the HERA cipher in terms of encryption speed and ciphertext expansion. The source codes of server-side computation are developed in Golang version 1.16.4 with `Lattigo` library [1] which implements RNS variants of the FV and CKKS schemes. The source codes of client-side computation are developed in C++17, using GNU C++ 7.5.0 compiler with AVX2 instruction set. XOF is instantiated with SHAKE256 in `XKCP` library [49]. Our experiments are done in AMD Ryzen 7 2700X @ 3.70 GHz single-threaded with 64 GB memory.

Additionally, we evaluate the performance of HERA combined with BGV only in order to make a fair comparison with previous works. One can find the result in the full version [19].

5.1 Parameter

The sets of parameters used in our implementation are given in Table 4, where

- λ is the security parameter;
- p is the number of precision bits of the RtF framework;
- L' is the ciphertext level at the end of the 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 the FV scheme in the RtF framework;
- QP is the largest ciphertext modulus of the HE schemes including special primes.

For the CKKS scheme, the message space is $\mathbb{C}^{\ell/2}$.

In Table 4, we recommend secure parameters of HERA when combined with the RtF framework. For the parameters related to bootstrapping, we follow the choice of bootstrapping parameters in [8]. Specifically,

- the hamming weight h of the secret key is 192;
- the range K of the sine evaluation is 25;
- the number R of the double angle formula is 2;
- the degree d_{sin} of the sine evaluation is 63;
- the degree d_{arcsin} of the inverse sine evaluation is 7, if necessary.

We also experiment the effect of the inverse sine evaluation [38]. The parameter names ending with `a` stands for the evaluation of the inverse sine function. The parameter sets ending with `s` stands for a small number of slots. It uses 16 slots in order to evaluate HERA. Parameter q_0/Δ' is the ratio of the first ciphertext modulus q_0 to the bootstrapping scaling factor Δ' which is introduced in Section 3.3. We use 128-bit secure HE parameters for all parameter sets. If an application requires more depth with 80-bit security, then a few dozens of level can be appended without raising N and degradation of security.

Set	λ	SKE		HE					
		$\lceil \log t \rceil$	r	$\log N$	$\log \ell$	$\lceil \log QP \rceil$	Δ_{CKKS}	q_0/Δ'	arcsin
`Par-80`	80	28	4	16	16	1533	2^{40}	512	✗
`Par-80a`	80	25	4	16	16	1533	2^{45}	16	✓
`Par-80s`	80	28	4	16	4	1533	2^{40}	512	✗
`Par-80as`	80	25	4	16	4	1533	2^{45}	16	✓
`Par-128`	128	28	5	16	16	1533	2^{40}	512	✗
`Par-128a`	128	25	5	16	16	1533	2^{45}	16	✓
`Par-128s`	128	28	5	16	4	1533	2^{40}	512	✗
`Par-128as`	128	25	5	16	4	1533	2^{45}	16	✓

Table 4: Selected sets of parameters used in our implementation. The rest of the bootstrapping parameters is set to be $(h, K, R, d_{\sin}, d_{\text{arcsin}}) = (192, 25, 2, 63, 0/7)$.

Set	CER	Client-side		Server-side			p	$\log q_{L'}$
		Lat. (μs)	Thrp. (MB/s)	Lat.		Thrp. (KB/s)		
				Off (s)	On (s)			
`Par-80`	1.54	1.520	22.86	98.56	16.84	5.066	17.22	500
`Par-80a`	1.24	1.443	26.62	91.09	20.68	5.412	19.13	375
`Par-80s`	1.53	1.520	22.95	71.89	13.23	0.0019	17.29	500
`Par-80as`	1.23	1.443	26.77	68.31	14.14	0.0020	19.25	375
`Par-128`	1.54	1.599	21.73	128.7	19.00	4.738	17.22	500
`Par-128a`	1.24	1.520	25.26	120.7	20.88	5.077	19.13	375
`Par-128s`	1.54	1.599	21.72	89.62	13.34	0.0018	17.21	500
`Par-128as`	1.23	1.520	25.26	84.02	14.31	0.0019	19.35	375

Table 5: Performance of the RtF transciphering framework with HERA.

5.2 Benchmarks

We measure the performance of the RtF 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 $\mathbb{Z}_t^{16}$), keystream generation from HERA, message scaling, rounding and vector addition over $\mathbb{Z}_t$. The extendable output function is instantiated with SHAKE256 in `XKCP`.

The server-side offline latency includes time for the randomized key schedule, homomorphic evaluation of the keystreams from HERA, and $\mathsf{SlotToCoeff}^{\mathsf{FV}}$. HERA is homomorphically evaluated by using row-wise packing. The online latency includes scaling up to FV-ciphertext space, the homomorphic subtraction, rescaling to the lowest level, and HalfBoot. We measure the latency until the first HE-ciphertext comes out, while the throughput is measured 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.

We summarize our implementation results in Table 5. This table includes ciphertext expansion ratio (CER), time-relevant measurements, precision, and homomorphic capacity. One can see that the parameters with arcsin (`Par-a`) offer smaller CER while the remaining levels are less than other parameters. On the other hand, the parameters with small slots (`Par-s`) take less time for evaluation since the complexity of evaluating $\mathsf{SlotToCoeff}^{\mathsf{FV}}$ and $\mathsf{CoeffToSlot}^{\mathsf{CKKS}}$ is affected by the number of slots.

Comparison. We compare the result to the recent implementation of LWEs-to-RLWE conversions [40] and CKKS itself. The comparison is summarized in Table 6. We run all those schemes by ourselves except the †-marked one. The source codes of LWEs-to-RLWE conversion is taken from the `OpenPegasus` library[7]. As `OpenPegasus` library does not include symmetric LWE encryption, we implement (seeded) symmetric LWE encryption with AVX2-optimized SHAKE256. We use `Lattigo` library for CKKS encryption.

In this table, the security parameter λ is set to 128. For the fairness of comparison, we try to make L' equal. Regardless of ℓ, the number of real number messages encrypted on the client side using RtF and LWE is 16 and 1 respectively. We evaluate the LWE encryption in LWEs-to-RLWE and the CKKS encryption in CKKS-only environment as (seeded) symmetric encryptions since they offer smaller ciphertext expansion ratio. For all experiments, we sample the domain of each component of the message vector from uniform distribution over $(-1, 1)$. When computing the ciphertext expansion ratio, we use the formula $\log t/(p+1)$, which excludes the effect of sending a public nonce. Multiple use of different nonces can be dealt with a counter so that the effect of nonce to the ratio is asymptotically zero.

Since the `OpenPegasus` library supports only selected sets of parameters in terms of the number of slots and the ciphertext modulus (at the point of submission), we implemented LWEs-to-RLWE for $N = 2^{16}$ and $\ell = 2^{10}$ which does not provide exactly the same functionality as ours with full available slots; we additionally implemented the RtF framework with HERA using the parameter $\ell = 2^9$ which processes the same number of data in order to make a fair comparison.

One can see that our RtF framework outperforms the LWEs-to-RLWE conversion and the CKKS-only environment with respect to CER and client-side performance, achieving the main purpose of the transciphering framework. On the server-side, the RtF framework enjoys larger throughput at the cost of larger latency due to the highly nonlinear structure of the HERA compared to LWE encryption. We note that the CKKS-only environment requires no additional computation since it uses CKKS-ciphertexts with nonzero level.

[7] `https://github.com/Alibaba-Gemini-Lab/OpenPEGASUS`

Scheme	N	ℓ	Ctxt. Exp.		Client		Server		p	L'
			Ctxt. (KB)	Ratio	Lat. (μs)	Thrp. (MB/s)	Lat. (s)	Thrp. (KB/s)		
RtF	2^{16}	2^{16}	0.055	1.54	1.599	21.73	147.68	4.738	17.2	11
RtF	2^{16}	2^{9}	0.055	1.53	1.599	21.78	117.71	0.051	17.2	11
LWE [40]	$2^{16}(2^{10})$	2^{10}	0.007	4.84	21.60	0.051	89.61	0.006	9.2	11
LWE [40]†	$2^{16}(2^{10})$	2^{13}	0.007	-	-	-	51.71	-	-	6
CKKS	2^{14}	2^{14}	640	35.31	14527	1.218	none		17.1	11

†: data is directly from the paper.

Table 6: Comparison of the RtF transciphering framework with HERA to LWEs-to-RLWE conversion (denoted by LWE) and the CKKS-only environment. All the experiments are done with 128-bit security. Parameter N in parentheses implies the dimension of LWE. The parameter p stands for the bits of precision. "-" indicates that the previous work did not report the value.

References

[1] Lattigo v2.1.1. Online: `http://github.com/ldsec/lattigo` (Dec 2020), ePFL-LDS

[2] 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)

[3] Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015. vol. 9056, pp. 430–454. Springer (2015)

[4] 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)

[5] Ashur, T., Dhooghe, S.: MARVELlous: a STARK-Friendly Family of Cryptographic Primitives. IACR Cryptology ePrint Archive, Report 2018/1098 (2018), `https://eprint.iacr.org/2018/1098`

[6] Baignères, T., Stern, J., Vaudenay, S.: Linear Cryptanalysis of Non Binary Ciphers. In: Adams, C., Miri, A., Wiener, M. (eds.) Selected Areas in Cryptography. vol. 4876, pp. 184–211. Springer (2007)

[7] 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)

[8] Bossuat, J.P., Mouchet, C., Troncoso-Pastoriza, J., Hubaux, J.P.: Efficient Bootstrapping for Approximate Homomorphic Encryption with Non-sparse Keys. In: Canteaut, A., Standaert, F.X. (eds.) Advances in Cryptology – EUROCRYPT 2021. pp. 587–617. Springer (2021)

[9] 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)

[10] 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)

[11] 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)

[12] Chen, H., Dai, W., Kim, M., Song, Y.: Efficient Homomorphic Conversion Between (Ring) LWE Ciphertexts. In: Sako, K., Tippenhauer, N.O. (eds.) Applied Cryptography and Network Security. pp. 460–479. Springer (2021)

[13] 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`

[14] 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)

[15] 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)

[16] 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)

[17] 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)

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

[19] Cho, J., Ha, J., Kim, S., Lee, B., Lee, J., Lee, J., Moon, D., Yoon, H.: Transciphering Framework for Approximate Homomorphic Encryption (Full Version). Cryptology ePrint Archive, Report 2020/1335 (2020), `https://eprint.iacr.org/2020/1335`

[20] 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)

[21] 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)

[22] 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)

[23] 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)

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

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

[26] Fröberg, R.: An Inequality for Hilbert Series of Graded Algebras. MATHEMATICA SCANDINAVICA **56** (Dec 1985)

[27] 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)

[28] 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)

[29] 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)

[30] 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)

[31] 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)

[32] 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)

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

[34] 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 (2020)

[35] 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)

[36] 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)

[37] 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)

[38] Lee, J.W., Lee, E., Lee, Y., Kim, Y.S., No, J.S.: High-Precision Bootstrapping of RNS-CKKS Homomorphic Encryption Using Optimal Minimax Polynomial Approximation and Inverse Sine Function. In: Canteaut, A., Standaert, F.X. (eds.) Advances in Cryptology – EUROCRYPT 2021. pp. 618–647. Springer (2021)

[39] 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)

[40] Lu, W., Huang, Z., Hong, C., Ma, Y., Qu, H.: PEGASUS: Bridging Polynomial and Non-polynomial Evaluations in Homomorphic Encryption. In: 2021 2021 IEEE Symposium on Security and Privacy (SP). pp. 1057–1073. IEEE Computer Society (may 2021)

[41] 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)

[42] 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)

[43] 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)

[44] 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)

[45] 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)

[46] 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)

[47] Rechberger, C., Soleimany, H., Tiessen, T.: Cryptanalysis of Low-Data Instances of Full LowMCv2. IACR Transactions on Symmetric Cryptology **2018**(3), 163–181 (2018)

[48] Regev, O.: On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. J. ACM **56**(6) (Sep 2009)

[49] XKCP: eXtended Keccak Code Package. `https://github.com/XKCP/XKCP` (Aug 2020)