FRAST: TFHE-friendly Cipher Based on Random S-boxes

Mingyu Cho*[1], Woohyuk Chung[2], Jincheol Ha[2], Jooyoung Lee[2], Eun-Gyeol Oh[2], and Mincheol Son[2]

[1] Mobilint, Inc., Seoul, Korea,
mingyu@mobilint.com
[2] KAIST, Daejeon, Korea,
{hephaistus,smilecjf,hicalf,eun-gyeol.oh,encrypted.def}@kaist.ac.kr

Abstract. A transciphering framework, also known as hybrid homomorphic encryption, is a practical method of combining a homomorphic encryption (HE) scheme with a symmetric cipher in the client-server model to reduce computational and communication overload on the client side. As a server homomorphically evaluates a symmetric cipher in this framework, new design rationales are required for "HE-friendly" ciphers that take into account the specific properties of the HE schemes.

In this paper, we propose a new TFHE-friendly cipher, dubbed FRAST, with a TFHE-friendly round function based on a random S-box to minimize the number of rounds. The round function of FRAST can be efficiently evaluated in TFHE by a new optimization technique, dubbed double blind rotation. Combined with our new WoP-PBS method, the double blind rotation allows computing multiple S-box calls in the round function of FRAST at the cost of a single S-box call. In this way, FRAST enjoys 2.768 (resp. 10.57) times higher throughput compared to Kreyvium (resp. Elisabeth) for TFHE keystream evaluation in the offline phase of the transciphering framework at the cost of slightly larger communication overload.

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

1 Introduction

Homomorphic Encryption. In order to achieve both privacy and usability of data in use, various types of homomorphic encryption (HE) schemes have been proposed and widely studied. HE schemes are typically targeted at the client-server model, where a large amount of data from clients is processed by an untrusted server. In this scenario, each client encrypts its data using an HE scheme and sends the HE-ciphertexts to the server so that the server is able to perform meaningful computations on the encrypted data without decrypting it.

* This work was done while M. Cho was a master's student at KAIST.

However, this approach of homomorphically encrypting all the clients' data in the client side has two technical problems. One is that a client should encrypt all its data with an HE scheme, which is heavier than traditional symmetric encryption schemes. Typically, even public key schemes such as RSA are only used for key-sharing and symmetric key schemes are used for actual data encryption in practice, so it would be desirable to reduce the computational cost required to the client with the help of the server. The other problem, which seems inevitable for HE schemes, is ciphertext expansion that increases the amount of data to transfer to the server.

Transciphering Framework. In order to address the problems mentioned above, a transciphering framework, also known as hybrid homomorphic encryption, has been proposed [45]. In this framework, the client encrypts its data using a symmetric key cipher, and the server homomorphically recovers the data to get the HE-ciphertext of the original data.

More precisely, the client chooses a symmetric key $\mathbf{k}$ of a symmetric cipher E, computes the HE-ciphertext of $\mathbf{k}$, namely, $\mathsf{Enc}^{\mathsf{HE}}(\mathbf{k})$ using an HE scheme HE, and sends it to the server only once. After that, when the client wants to send data $\mathbf{m}$ to the server, the client encrypts $\mathbf{m}$ using E with key $\mathbf{k}$ and sends the ciphertext $\mathbf{c}$ to the server. Then the server homomorphically encrypts it to obtain $\mathsf{Enc}^{\mathsf{HE}}(\mathbf{c})$, and then homomorphically evaluates the decryption circuit of E using $\mathsf{Enc}^{\mathsf{HE}}(\mathbf{k})$, obtaining the HE-ciphertext $\mathsf{Enc}^{\mathsf{HE}}(\mathbf{m})$ of the original data $\mathbf{m}$. Figure 1 describes the transciphering framework using a stream cipher, where the server precomputes $\mathsf{Enc}^{\mathsf{HE}}(\mathbf{z})$ for a keystream $\mathbf{z}$ from the stream cipher in the offline phase, and $\mathsf{Enc}^{\mathsf{HE}}(\mathbf{m})$ is obtained by homomorphically computing $\mathsf{Enc}^{\mathsf{HE}}(\mathbf{c}) - \mathsf{Enc}^{\mathsf{HE}}(\mathbf{z})$ in the online phase. In the transciphering framework, the client takes the following advantages.

- A client does not need to encrypt all its data using an HE scheme (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 lower 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. For most HE schemes, linear operations such as addition and constant multiplication are way cheaper than nonlinear operations such as multiplication. With this observation, the efficiency of an HE-friendly cipher has been known to depend on its nonlinear operations. That said, more specific design rationale might differ according to the specific properties of the HE scheme. When combined with BGV [16] and BFV [15, 29], HE schemes supporting homomorphic addition and multiplication over finite fields, it would be desirable to use as few multiplications as possible. However, multiplication

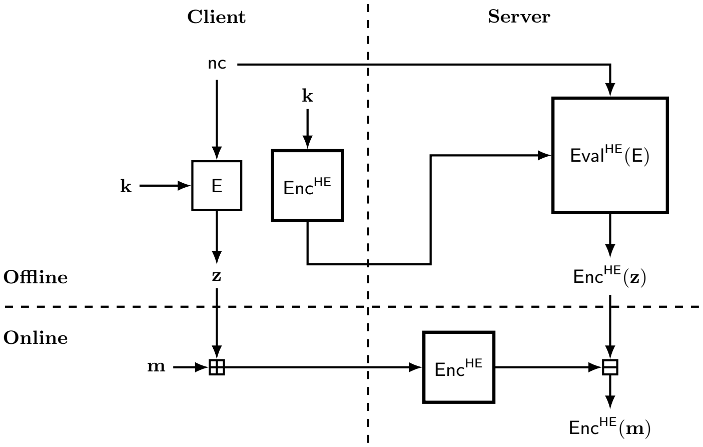


Fig. 1: The transciphering framework with a stream cipher. Homomorphic operations are performed in the boxes with thick lines. The vertical dashed line distinguishes the client and server, and the horizontal dashed line distinguishes the offline and online computations. The encryption $\mathsf{Enc}^{\mathsf{HE}}$ performed by the server in the online phase can be done trivially.

is needed to construct nonlinear layers, which are essential to achieve security against various attacks including differential, linear, and algebraic attacks. So designing a symmetric cipher with simple nonlinear layers might require a large number of rounds. To address this issue, a novel design strategy has been proposed [43]: using randomly generated linear layers and simple nonlinear layers. Many HE-friendly ciphers (for the BGV and BFV schemes) follow this design rationale.

However, when it comes to the TFHE scheme [20, 21], one might take a different strategy of designing TFHE-friendly ciphers due to the difference of TFHE from BGV and BFV. We note that when TFHE [20] was first proposed, it supported homomorphic XOR and AND operations on its ciphertexts encrypting a single bit of message, both of which require the same computational cost of one bootstrapping. Later, the leveled version of TFHE [21] was proposed[3] by expanding the plaintext encoding to $\mathbb{Z}_{2^t}$ for a small integer t such as $t = 4$. In particular, it enjoys fast linear operations and nonlinear operations by means of table lookup. In this paper, TFHE refers to the leveled version of TFHE.

The table lookup operation, homomorphically performed by programmable bootstrapping (PBS), is a distinctive feature of TFHE. Since the table lookup cost does not depend on the algebraic representation of a function to evaluate, it is not required to approximate nonlinear functions into polynomials, as in the case of BGV and BFV. Applying PBS to nonlinear operations such as ReLU and max, [21] enables efficient homomorphic inference of deep neural networks.

[3] It was named Concrete in [21], while recently it is also simply called TFHE.

In terms of designing TFHE-friendly ciphers, homomorphic nonlinear operation by PBS removes the requirement of the algebraic simplicity of the nonlinear layers. On the other hand, programmable bootstrapping requires that either one padding bit is empty in the most significant bit (MSB) of the value hidden in the ciphertext[4], or the table is negacyclic. For example, the Elisabeth cipher [25], first proposed as a TFHE-friendly cipher, uses 4-bit negacyclic S-boxes in the nonlinear layers.

Regardless of which HE scheme is used, an HE-friendly cipher is typically designed as a stream cipher. The authors of Kreyvium [17], a variant of Trivium [27] with 128-bit security, first claimed that with a stream cipher, a keystream can be precomputed independently of a message in the transciphering framework, leading to simple homomorphic decryption in the online phase[5]. Since then, most HE-friendly ciphers have been designed as stream ciphers. As the online phase can be made simple with a stream cipher, high throughput for keystream evaluation in the offline phase becomes of practical relevance, in particular, in an environment where a large amount of data is transferred.

1.1 Our Contribution

In this paper, we propose a new TFHE-friendly stream cipher, dubbed FRAST (Feistel with RAndomly generated S-boxes for Tfhe), which enjoys high throughput for its keystream evaluation on TFHE. The main design rationale of FRAST is twofold.

First, the round function is designed to allow multiple S-box evaluations at the cost of almost a single S-box evaluation. More precisely, it computes $S(x+rk_i)$ for a common input x and multiple round keys rk_i for an S-box used in the round function. Since the inputs to the S-box are distinct, a naive way of evaluating the round function of FRAST would be to evaluate each $S(x+rk_i)$ separately.

On the other hand, in order to optimize the TFHE evaluation of FRAST, we opt for the structure such that the inputs to the S-box share a common value x, while the round keys rk_i's are fixed once the master key is chosen. Then $S(x+rk_i)$ can be evaluated for multiple round keys rk_i by sharing the internal state of computing $S(x)$ at the cost of some precomputation on the round keys. We call this type of optimization technique *double blind rotation*.

As mentioned before, the PBS operation basically requires either one empty padding bit in the ciphertext or the negacyclicity of the function to evaluate. To address this issue, advanced PBS techniques called programmable bootstrapping without padding (WoP-PBS)[6] are proposed [22, 8, 49, 40, 39, 23]. Since FRAST uses a non-negacyclic S-box for its component, we also propose a new WoP-PBS method supporting the double blind rotation.

[4] Precisely, the MSB should be known to perform PBS correctly.

[5] In the case of TFHE, one PBS operation might be required to clear a carry bit after the subtraction. See Section 6 for details.

[6] It is also called the full domain functional bootstrapping (FDFB).

Our WoP-PBS uses three GenPBS operations in a naive evaluation without requiring additional evaluation keys or larger TFHE parameters for a larger PBS precision. To the best of our knowledge, ComBo [23] is the only WoP-PBS method satisfying the above constraints, while it uses four GenPBS operations in its naive evaluation. Combined with the multi-value PBS [18] or PBSmanyLUT [22], it is possible to reduce one GenPBS for both our WoP-PBS and ComBo. Using parallel computation with $\log p$ threads, the latency of our WoP-PBS becomes almost the same as a single GenPBS, where the plaintext space is $\mathbb{Z}_p$ for some power-of-two p. For ComBo, the latency can be reduced to two GenPBS operations using parallel computation with two threads.

The second feature of our design is that the round function is based on randomly generated S-boxes for some rounds. We note that the TFHE evaluation of an S-box is independent of its structure unless it is constant or negacyclic. Exploiting this property of the TFHE operation, random S-boxes efficiently mitigate various attacks using multiple input-output pairs from a fixed function. On the other hand, some rounds of FRAST are still based on fixed S-boxes to guarantee concrete security against algebraic attacks.

We implement FRAST using the `tfhe-rs` library [50]. FRAST achieves 2.768 (resp. 10.57) times higher throughput compared to Kreyvium (resp. Elisabeth) on the server-side offline phase.

1.2 Related Work

In this section, we briefly review some existing TFHE-friendly ciphers. The FLIP stream cipher [43] is based on a filter permutator that randomly permutes key bits and computes a nonlinear function, called a filter function, generating a single keystream bit. The filter function is chosen to have a simple algebraic representation for its efficient homomorphic evaluation, and its low security is enhanced by the randomly generated permutation layer. On the other hand, the permutation layer is publicly generated, so one can homomorphically evaluate this layer with almost no cost.

Later, an improved filter permutator, dubbed FiLIP, has been proposed [42]. As both FLIP and FiLIP have been proposed for the 3rd generation FHE such as TFHE, Hoffmann et al. [36] proposed an efficient evaluation of FiLIP with TFHE. Cong et al. [24] adopt a transciphering using the FiLIP cipher in private decision tree evaluation, but they use the FINAL [12] scheme for the transciphering, which is an HE scheme based on the NTRU assumption. Méaux et al. [44] also proposed a FINAL-based transciphering with FiLIP that supports setup-independent plaintext space.

The (improved) filter permutator originally works on the Boolean space $\mathbb{F}_2$. By generalizing this space to a group, Cosseron et al. [25] proposed a TFHE-friendly cipher Elisabeth. In this construction, $\mathbb{Z}_{16}$-addition is used for the group operation and the filter function consists of $\mathbb{Z}_{16}$-addition and evaluation of some negacyclic S-boxes. This exploits the negacyclic-friendly property of the PBS operation in TFHE while its overall structure still follows the strategy of randomly generating its permutation layer. Recently, the Elisabeth cipher has been broken by an algebraic attack [34]. After that, Hoffmann et al. [37] proposed several patches for Elisabeth, named Elisabeth-b, Gabriel and Margrethe, whose TFHE evaluation cost is at least twice more than Elisabeth under a single thread.

For the other type of TFHE-friendly ciphers, Balenbois et al. [4] proposed to use Trivium and Kreyvium in the transciphering framework with TFHE. Kreyvium is a variant of a stream cipher Trivium of 80-bit security, that supports a larger key to achieve 128-bit security. Once an initial vector IV is chosen, the key and IV are loaded on the registers. The registers are updated by a nonlinear function, which also generates keystream bits after some initialization rounds. They presented an efficient TFHE evaluation of Trivium and Kreyvium keystreams by the multithreading technique.

2 Preliminaries

2.1 Notations

Throughout the paper, bold lowercase letters (resp. bold uppercase letters) denote vectors (resp. matrices). For two vectors (bit strings) $\mathbf{a}$ and $\mathbf{b}$, their concatenation is denoted by $\mathbf{a}\|\mathbf{b}$. $\lfloor r \rceil$ denotes the nearest integer to r, rounding upwards in case of a tie. A real interval $[a, b)$ has an alternative notation: $[a, b[$. For two integers a and b, $\mathbb{Z} \cap [a, b[$ is denoted by $[\![a, b[\![$. For an integer q, we identify $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ with $[\![-q/2, q/2[\![$ in the context of TFHE. The set $\mathbb{B}$ and $[n]$ denote $\{0, 1\}$ and $\{1, 2, \ldots, n\}$, respectively, for a positive integer n. 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 $\mathcal{D}$, $a \leftarrow \mathcal{D}$ denotes that a is sampled according to the distribution $\mathcal{D}$. Unless stated otherwise, all logarithms are to the base 2.

In the context of TFHE, we use p and q for the moduli of messages and ciphertexts, respectively. We only consider the case where p and q are powers of two. For a power-of-two N, we denote the cyclotomic ring $\mathbb{Z}[X]/(X^N + 1)$ by $\mathbb{Z}_N[X]$. For the polynomial ring over $\mathbb{Z}_q$, we write $\mathcal{R}_{q,N} = \mathbb{Z}_q[X]/(X^N + 1)$. Similarly, we write $\mathbb{B}_N[X] = \mathbb{B}[X]/(X^N + 1)$.

2.2 TFHE

In this section, we briefly review the core concepts of the TFHE scheme. Although TFHE itself is mathematically defined over the real torus $\mathbb{T} = \mathbb{R}/\mathbb{Z}$ [20], it is common to use the discretized torus $\frac{1}{q}\mathbb{Z}/\mathbb{Z}$ for $q = 2^{32}$ or $q = 2^{64}$ considering its implementation. Hence, we identify the discretized torus $\frac{1}{q}\mathbb{Z}/\mathbb{Z}$ as $\mathbb{Z}_q$, which is commonly used in the recent descriptions of TFHE [21, 22, 8].

LWE, RLWE, and GLWE Ciphertexts. Under a secret key $\mathbf{S} \in \mathcal{R}_{q,N}^k$, a message $M \in \mathcal{R}_{p,N}$ is encrypted into a GLWE ciphertext $\mathbf{c} \in \mathcal{R}_{q,N}^{k+1}$ with a scaling factor Δ such that $\Delta \leq q/p$ as follows [16].

$$\mathbf{c} = \mathrm{GLWE}_{\mathbf{S}}(\Delta \cdot M) = (A_1, \ldots, A_k, B = \sum_{i=1}^{k} A_i \cdot S_i + [M \cdot \Delta]_q + E)$$

where $\mathbf{S} = (S_1, \ldots, S_k)$, $A_i \leftarrow \mathcal{R}_{q,N}$ for $i = 1, 2, \ldots, k$, and $E \leftarrow \chi_\sigma$ for some Gaussian distribution χ_σ denoting the error distribution. $(A_1, \ldots, A_k)$ and B are called the mask and the body of the GLWE ciphertext $\mathbf{c}$, respectively, and k is called the GLWE dimension. It is common to use a binary secret key in the TFHE scheme, so we only deal with binary secret keys in this paper.

A GLWE ciphertext with $N = 1$ is called an LWE ciphertext. In this case, it is common to use n to denote the LWE dimension instead of k, so that an LWE ciphertext is usually denoted $(a_1, \ldots, a_n, b) \in \mathbb{Z}_q^{n+1}$. When $k = 1$, a GLWE ciphertext is called an RLWE ciphertext. In this paper, we refer LWE ciphertexts separately from GLWE ciphertexts of $N > 1$.

The decryption of a GLWE ciphertext is computing its phase, which is defined as $B - \langle (A_1, \ldots, A_k), \mathbf{S} \rangle$, followed by rounding the phase by the scaling factor $\varDelta$. The decryption works correctly if the error contained in the ciphertext is small enough to be eliminated during the rounding by $\varDelta$.

From the definition of the GLWE ciphertext, the addition of two GLWE ciphertexts under the same secret key results in the addition of their internal plaintexts in $\mathcal{R}_{q,N}$. Multiplying the ciphertext by a scalar plaintext is possible by iterating the addition several times. Both the addition and the scalar multiplication increase the error of the resulting ciphertext linearly.

GGSW Ciphertexts. In the case of nonlinear operations such as multiplication, TFHE uses another type of ciphertext called GGSW [33]. Let $B \in \mathbb{N}$ be a power-of-two and $\ell \in \mathbb{N}$. A GGSW ciphertext $\mathbf{C} \in \mathcal{R}_{q,N}^{\ell(k+1)\times(k+1)}$ of a message $M \in \mathbb{Z}_N[X]$ under a secret key $\mathbf{S} \in \mathbb{B}_N[X]^k$ is an $\ell(k+1) \times (k+1)$ matrix over $\mathcal{R}_{q,N}$ defined as follows.

$$\mathbf{C} = \left(\mathrm{GLWE}_{\mathbf{S}} \left(-S_i \cdot \frac{q}{B^j} M \right) \right)_{(i,j)\in[k+1]\times[\ell]}$$

where $\mathbf{S} = (S_1, \ldots, S_k)$, $S_{k+1} = -1$, and each GLWE ciphertext is considered as a row having $k + 1$ columns of polynomials in $\mathcal{R}_{q,N}$. B and ℓ are called the decomposition base and the decomposition level of the GGSW ciphertext $\mathbf{C}$, respectively.

External Product and CMux Gate. The external product $\boxdot$ between GGSW ciphertext $\mathbf{C}$ and GLWE ciphertext $\mathbf{c}$ is defined as

$$\mathbf{C} \boxdot \mathbf{c} = \mathrm{GadgetDecomp}(c) \cdot \mathbf{C}$$

where $\mathrm{GadgetDecomp}(\mathbf{c}) \in \mathcal{R}_{q,N}^{\ell(k+1)}$ is the gadget decomposition of $\mathbf{c}$ [33, 22] of which coefficients are lying in $[\![-B/2, B/2[\![$. The external product between GGSW and GLWE ciphertexts defines homomorphic module scalar multiplication on the discretized torus $\frac{1}{q}\mathbb{Z}/\mathbb{Z}$. Roughly speaking, the external product increases the error by the magnitude of the plaintext in the GGSW ciphertext. Thus it is common to use GGSW ciphertext encrypting a single bit of message in the external product.

The controlled mux gate, dubbed CMux, is the key operation used in TFHE. Suppose two GLWE ciphertexts $\mathbf{c}_0$ and $\mathbf{c}_1$ are given along with a secret Boolean value b encrypted to a GGSW ciphertext $\mathbf{C}$, where all three ciphertexts are encrypted with the same key $\mathbf{S}$. Then one may select $\mathbf{c}_b$ without knowing b by

$$\mathrm{CMux}(\mathbf{C}, \mathbf{c}_0, \mathbf{c}_1) = (\mathbf{c}_1 - \mathbf{c}_0) \boxdot \mathbf{C} + \mathbf{c}_0.$$

Programmable Bootstrapping. The programmable bootstrapping (PBS) of TFHE supports an extra functionality that evaluates a function for free during the bootstrapping. Suppose an LWE ciphertext $\mathbf{c} = (a_1, \dots, a_n, b) \in \mathbb{Z}_q^{n+1}$ of a message m under a secret key $\mathbf{s} = (s_1, \dots, s_n) \in \mathbb{B}^n$ is given. The PBS operation outputs a refreshed LWE ciphertext $\mathbf{c}' \in \mathbb{Z}_q^{kN}$ of the message $f(m)$ under a secret key $\mathbf{s}' \in \mathbb{B}^{kN}$ by the following steps.

1. Encode the function f on a new GLWE ciphertext under a different secret key $\mathbf{S}' \in \mathbb{B}_N[X]^k$. The half of the function values of f are redundantly encoded in the coefficients of the plaintext of the (trivial) GLWE ciphertext.
2. (Modulus switch) Compute $\tilde{\mathbf{c}} = (\tilde{a}_1, \dots, \tilde{a}_n, \tilde{b}) \in \mathbb{Z}_{2N}^{n+1}$ where

   $$\tilde{a}_i = \lfloor a_i \cdot (2N)/q \rceil \text{ and } \tilde{b} = \lfloor b \cdot (2N)/q \rceil,$$

   obtaining an LWE ciphertext encrypting $\tilde{m} \approx \lfloor m \cdot (2N)/q \rceil$.
3. (Blind rotation) Multiply $X^{-\tilde{b}+\sum_{i=1}^{n} \tilde{a}_i s_i} = X^{-\tilde{m}}$ to the GLWE ciphertext encoding the function using a bootstrapping key $\{\mathrm{GGSW}_{\mathbf{S}'}(s_i)\}_{i=1}^{n}$; multiply either 1 or $X^{-\tilde{a}_i}$ according to $s_i \in \{0, 1\}$ by the CMux gate.
4. (Sample extraction) Extract the constant term of the GLWE ciphertext, obtaining an LWE ciphertext of $f(m)$ under the secret key $\mathbf{s}' \in \mathbb{B}^{kN}$ which is a reordering of the coefficients of $\mathbf{S}'$.

In this paper, we call the above PBS operation GenPBS as in [22].

The requirement of negacyclicity of f comes from the modulus switching step; it computes $\tilde{m} \approx \lfloor m \cdot (2N)/q \rceil$ while $X^N = -1$. Hence, it is only possible to evaluate a negacyclic function $f : \mathbb{Z}_p \to \mathbb{Z}_q$ such that $f(x + p/2) = -f(x)$ by encoding only half of the function values. To evaluate an arbitrary function, TFHE requires one padding bit of zero in the MSB of m to guarantee $\tilde{m} < N$.

In this paper, we define the precision of GenPBS as the log of the number of the function values encoded in the GLWE ciphertext that the GenPBS operation evaluates. Then, a GenPBS operation of t-bit precision can compute an arbitrary function of t-bit precision, i.e., a function on $[\![0, 2^t[\![$, or an arbitrary negacyclic function of $(t + 1)$-bit precision without the padding bit. A GenPBS operation with a larger precision requires a larger polynomial size N, increasing computational cost for it.

Keyswitching. Since GenPBS outputs an LWE ciphertext under a different secret key, it has to be switched back to a ciphertext under the original secret key. This step is called the *keyswitching*. In practice, the keyswitching operation is performed only once just before the GenPBS operation to match the LWE dimension rather than after every GenPBS operation. To denote the keyswitching operation before the GenPBS operation, we denote a GenPBS after a keyswitching as KSthenGenPBS.

Plaintext Encoding in TFHE. To keep the padding bit zero, Bergerat et al. [8] proposed a new encoding method for TFHE splitting the traditional plaintext space into three parts: one (or more) bit of padding at the MSB, the carry subspace after the padding bits, and the message subspace at the LSBs. By tracking the maximum possible value in the ciphertext, it clears the carry space before the padding bit is filled. For example, the default parameters of `tfhe-rs` library for `shortint` type, called `PARAM_MESSAGE_2_CARRY_2`, uses the encoding that consists of two message bits, two carry bits, and one padding bit.

2.3 Algebraic Attacks

In this section, we briefly review algebraic attacks applicable to FRAST.

2.3.1 Trivial Linearization and Extended Linearization

Trivial Linearization. Given input-output pairs of a symmetric cipher, it is possible to construct a system of polynomial equations with respect to the key variables $\mathbf{k}$. Trivial linearization is to make the system linear by introducing new variables for all monomials of degree greater than one and to solve it. For the linearized system to be solved, the number of equations should be greater than or equal to the number of variables including newly introduced ones.

If a system of Boolean equations in n variables is of degree d and almost all the monomials of degree up to d appear in the system, then the complexity of the trivial linearization attack is given by

$$\left(\sum_{i=1}^{d}\binom{n}{i}\right)^{\omega}$$

ignoring the constant factor, where $2 \leq \omega \leq 3$ denotes the linear algebra constant.

Extended Linearization. Courtois et al. [26] proposed the eXtended Linearization algorithm that can be used when the number of equations is less than the number of monomials. Given a system of m equations of degree d in n variables over $\mathbb{F}_2$, the XL algorithm extends the system by multiplying all the monomials of degree at most $D - d$ for some $D(> d)$ to obtain a larger number of (linearly independent) equations of degrees at most D. As the number of equations grows faster than the number of monomials, it is possible to solve the system for a sufficiently large D. The problem is that it is hard to determine the smallest D, called the *solving degree*.[7]

When designing a symmetric cipher, we can assume that all the resulting equations are linearly independent, which is in favor of an adversary. Then it is

[7] The recent results show that the solving degree is the same as the degree of regularity [47, 2].

possible to estimate the solving degree D as the smallest one satisfying

$$\left(\sum_{i=0}^{D-d}\binom{n}{i}\right)m \geq \sum_{i=1}^{D}\binom{n}{i}$$

assuming that all the monomials appear in the extended system of equations. Then the complexity of the XL algorithm is given by

$$\left(\sum_{i=1}^{D}\binom{n}{i}\right)^{\omega} \tag{1}$$

ignoring the constant factor. There are some optimized variants such as the Wiedemann XL algorithm [48], while using (1) with $\omega = 2$ still gives a lower bound on its complexity.

Hybrid Strategy. A hybrid strategy that guesses the values of some variables can be applied to the linearization attacks. The complexity of the hybrid trivial linearization after guessing k variables is given by

$$\min_{k} 2^{k}\left(\sum_{i=1}^{d}\binom{n-k}{i}\right)^{\omega} \tag{2}$$

ignoring the constant factor.

For the XL attack, the solving degree might differ according to the number of guessed variables. Let $D_{n,k}$ be the solving degree of the system after guessing k variables. Then the complexity of the hybrid XL attack over $\mathbb{F}_2$ is given by

$$\min_{k} 2^{k}\left(\sum_{i=1}^{D_{n,k}}\binom{n-k}{i}\right)^{\omega} \tag{3}$$

ignoring the constant factor.

2.3.2 Gröbner Basis Attack

The Gröbner basis attack is to solve a system of equations by computing its Gröbner basis. The complexity of Gröbner basis computation can be estimated using the *degree of regularity* of the system of equations [5], upper bounding the degree of polynomials that occur during the computation of a Gröbner basis using algorithms such as F_4 [30] and F_5 [31].

Consider a system $\{f_i\}_{i=1}^{m}$ of m equations in n variables. Let d_i denote the degree of f_i for $i = 1, 2, \ldots, m$. When working on $\mathbb{F}_2$, Bardet et al. [6] proposed the following Hilbert series to estimate the degree of regularity taking homogenization into account[8].

$$T_{m,n}(z) = \frac{(1+z)^{n}}{(1-z)\prod_{i=1}^{m}(1+z^{d_i})}. \tag{4}$$

[8] It already takes into account the field equations of the form $x^2 - x = 0$ over $\mathbb{F}_2$.

The degree of regularity d_{reg} is determined by the smallest degree of the term with a non-positive coefficient in the series $T_{m,n}$. Given the degree of regularity d_{reg} over $\mathbb{F}_2$, the complexity of computing a Gröbner basis is known to be

$$O\left(\binom{n}{d_{\mathsf{reg}}}^{\omega}\right). \tag{5}$$

The degree of regularity estimated by the Hilbert series is commonly used to theoretically estimate the complexity of the Gröbner basis attack. However, it requires an assumption that the system of equations is semi-regular. It is known that almost all polynomial sequences are semi-regular [32], but for a system obtained from a symmetric cipher it might not be the case since it has a certain structure.

3 The FRAST Cipher

3.1 Specification

A stream cipher FRAST with 128-bit security takes as input a 256-bit key $\mathbf{k} \in \mathbb{Z}_{16}^{64}$ and a 128-bit nonce $\mathsf{nc} \in \{0,1\}^{128}$, and returns a 128-bit keystream block $\mathbf{k}_{\mathsf{nc}} \in \mathbb{Z}_{16}^{32}$. The FRAST cipher has two types of round functions: the randomized one and the fixed one. Both types of round functions have the same structure except for the underlying S-boxes.

In a nutshell, FRAST consists of 40 rounds, namely,

$$\mathsf{FRAST}[\mathbf{k}, \mathsf{nc}] = \mathsf{RF}[\mathbf{k}, \mathsf{nc}, 40] \circ \mathsf{RF}[\mathbf{k}, \mathsf{nc}, 39] \circ \cdots \circ \mathsf{RF}[\mathbf{k}, \mathsf{nc}, 1]$$

where $\mathsf{RF}[\mathbf{k}, \mathsf{nc}, i]$ is the i-th round function using secret key $\mathbf{k}$ and nonce nc. For $(x_1, \ldots, x_{32}) \in \mathbb{Z}_{16}^{32}$, $\mathsf{RF}[\mathbf{k}, \mathsf{nc}, i](x_1, \ldots, x_{32}) = (y_1, \ldots, y_{32}) \in \mathbb{Z}_{16}^{32}$ is defined as

$$y_j = x_j + S_{\mathsf{erf}}^{(\mathsf{nc};\, i)}(x_1 + rk_j^{(i)}) \text{ for } j = 2, 3, \ldots, 32,$$
$$y_1 = x_1 + S_{\mathsf{crf}}^{(\mathsf{nc};\, i)}(y_2 + y_3 + \cdots + y_{32} + rk_1^{(i)})$$

where $\mathbf{rk}^{(i)} = (rk_1^{(i)}, rk_2^{(i)}, \ldots, rk_{32}^{(i)}) \in \mathbb{Z}_{16}^{32}$ is the i-th round key (derived from the master key $\mathbf{k}$), and $S_{\mathsf{erf}}^{(\mathsf{nc};\, i)}$ and $S_{\mathsf{crf}}^{(\mathsf{nc};\, i)}$ are the S-boxes used in the i-th round function[9] (see Figure 2).

FRAST repeats 4 random round functions followed by 1 fixed round function. Hence, the i-th round is a random round if i is not a multiple of 5, and is a fixed round if i is a multiple of 5.

S-Box. The random rounds of FRAST generate their S-boxes $S_{\mathsf{erf}}^{(\mathsf{nc};\, i)}$ and $S_{\mathsf{crf}}^{(\mathsf{nc};\, i)}$ as negacyclic functions; the first 8 function values $(S_{\mathsf{erf}}^{(\mathsf{nc};\, i)}(0), \ldots, S_{\mathsf{erf}}^{(\mathsf{nc};\, i)}(7))$ and

[9] The subscript erf (resp. crf) stands for expanding round function (resp. contracting round function), a term used in the generalized Feistel networks [1].

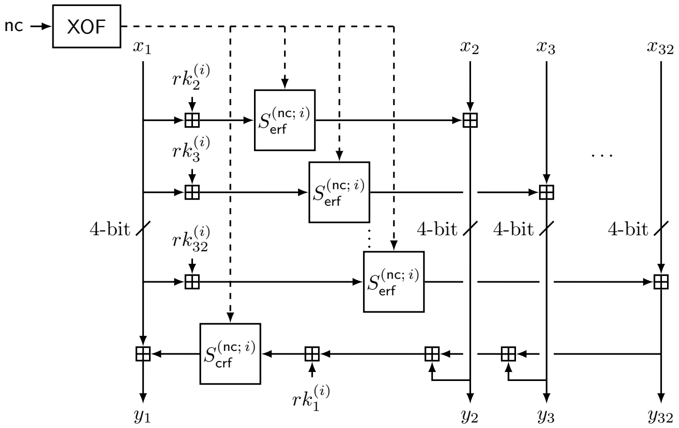


Fig. 2: The i-th round function of FRAST. $rk_1^{(i)}, \dots, rk_{32}^{(i)}$ are the i-th round keys.

$(S_{\mathsf{crf}}^{(\mathsf{nc};\,i)}(0), \dots, S_{\mathsf{crf}}^{(\mathsf{nc};\,i)}(7))$ are sampled by the output from the underlying extendable output function XOF with input nc, and the other function values are determined by the negacyclic property of the S-boxes.

The fixed rounds of FRAST use the same fixed S-box defined in Table 1 for their S-boxes $S_{\mathsf{erf}}^{(\mathsf{nc};\,i)}$ and $S_{\mathsf{crf}}^{(\mathsf{nc};\,i)}$. The S-box is one of the golden S-boxes of size 4-bit proposed in [46].

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$S(x)$	0	3	5	8	6	9	12	7	13	10	14	4	1	15	11	2

Table 1: The fixed S-box used in the fixed rounds of FRAST.

Key Schedule. The round keys $\mathbf{rk}^{(i)}$ for $i = 1, 2, \dots, 40$ are obtained by multiplying 64×64 invertible matrices over $\mathbb{Z}_{16}$ to the master key $\mathbf{k} \in \mathbb{Z}_{16}^{64}$. More precisely, an invertible matrix $\mathbf{M}$ is chosen from the output of $\mathsf{SHAKE256}$ with a fixed public input[10]. Then, round keys $\mathbf{rk}_{2i-1}$ and $\mathbf{rk}_{2i}$ are defined by $\mathbf{M}^i \cdot \mathbf{k} = \mathbf{rk}_{2i-1} \| \mathbf{rk}_{2i}$ for $i = 1, 2, \dots, 20$. The exact specification of $\mathbf{M}$ is given in Supplementary Material A.

Encryption Mode. When a keystream of m blocks in $(\mathbb{Z}_{16}^{32})^m$ is needed for some $m > 0$, the "inner-counter mode" can be used: for $\mathsf{ctr} = 0, 1, \dots, m-1$,

[10] We used the string `Feistel with RAndomly generated S-boxes for Tfhe` in our implementation, obtaining an invertible matrix over both $\mathbb{Z}_{16}$ and $\mathbb{F}_{2^4}$ without rejection.

one computes

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

where **ic** denotes a constant $(0, 1, \ldots, 15, 0, 1, \ldots, 15) \in \mathbb{Z}_{16}^{32}$. Figure 3 shows the overall structure of FRAST in the counter mode.

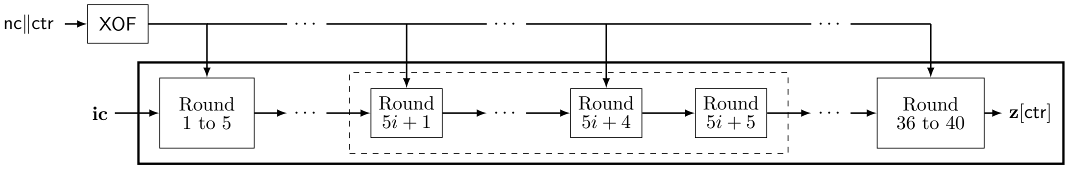


Fig. 3: The overall structure of FRAST in the counter mode, where **ic** is the public input constant and $\mathbf{z}[\mathsf{ctr}]$ is the keystream. Homomorphic operations are performed in the box with thick lines. The i-th round is a fixed round if i is a multiple of 5, and a random round otherwise.

3.2 Design Rationale

Each round of FRAST uses a common input branch to the S-boxes, while each S-box input is masked with a distinct round key. The motivation for using the common input branch lies in the multi-value PBS [18] that evaluates multiple functions on the same input at the cost of almost one PBS call. To evaluate multiple functions efficiently, it computes a common state by the blind rotation, the most expensive part of the PBS operation, and induces the desired function values by polynomial multiplications. Similarly to this method, by sharing the result of the blind rotation on the common input value and using the precomputed GGSW ciphertexts of the round key bits, we can evaluate multiple S-box calls on a common input with multiple round keys added at the cost of almost one S-box evaluation. This optimization technique is dubbed *double blind rotation* since one blind rotation by the common input is followed by another by the round key (see Section 4.2 for the details).

The round function of FRAST, which can be seen as a generalized Feistel network, has an issue on the error growth in its TFHE evaluation. Unlike the substitution-permutation network, the branches are not refreshed during the evaluation of S-boxes, so linear layers, linearly increasing the internal noise, are not "free" anymore. With a simple permutation layer, the number of rounds might be much higher than the number of branches due to the weak diffusion of the permutation layer.

It is a common strategy to design HE-friendly ciphers with random linear layers (in the substitution-permutation network) in order to reduce the required number of rounds. However, TFHE evaluation of such linear layers will lead to significant error growth. So we opt for random S-box generation to randomize the

FRAST encryption function. Since TFHE evaluates S-boxes by PBS operations, the random structure of the S-box will not affect the evaluation cost of FRAST.

The size of the S-boxes in FRAST is 4 bits, which is common in symmetric ciphers. As the performance of PBS operation with 4-bit precision is sufficiently efficient[11], we choose 4-bit S-boxes for FRAST. For the efficiency of the homomorphic S-box evaluation, the random S-boxes are defined by negacyclic functions whose half of the function values are sampled uniformly at random from $\mathbb{Z}_{16}$, determining the others by the negacyclic property.

On the other hand, the fixed S-box of FRAST is non-negacyclic. It is one of the golden S-boxes proposed in [46], which is known to have no polynomial representation over $\mathbb{Z}_{16}$ (see Lemma 1). Although FRAST uses addition over $\mathbb{Z}_{16}$, we analyze its security against algebraic attack with the XOR-variant of FRAST as done in the Elisabeth cipher, focusing on classical algebraic properties over $\mathbb{F}_2$. Over $\mathbb{F}_2$, all the output bits of the S-box have algebraic degree 3 and all the input bits affect the output bits nonlinearly. Its inverse also has the same algebraic properties. The number of quadratic equations induced by the S-box is 21, which is the minimum for 4-bit S-boxes. The algebraic representations of the S-box are given in Supplementary Material B. Zhang et al. [51] argue that the golden S-boxes proposed in [46] might be vulnerable to differential and linear attacks due to certain properties of the linear layers combined with the S-boxes. When it comes to FRAST, randomly generated S-boxes mitigate such vulnerabilities.

FRAST uses 256-bit keys, providing 128-bit security. The key length typically affects the communication overload of the transciphering since it should be sent to the server as a TFHE-ciphertext. When it comes to FRAST, most of the round keys are sent to the server to enable the double blind rotation, so that the length of the master key itself does not affect the communication overload. Hence, to achieve stronger security against algebraic attacks, FRAST uses master keys longer than the target level of security.

4 PBS Techniques for FRAST

In the transciphering framework with TFHE, each S-box of the cipher should be evaluated by the GenPBS operation. However, it requires either one bit of padding in the input ciphertext or the S-box to be negacyclic. Although the random S-boxes of FRAST are negacyclic, the fixed S-box of FRAST is non-negacyclic.

To address the issue of padding bit, PBS techniques called without-padding PBS (WoP-PBS) have been proposed [22, 49, 40, 8, 39]. Most of the previous WoP-PBS methods require either additional evaluation keys, such as LWE-toGLWE keyswitching keys [22, 39] and circuit bootstrapping keys [8], or parameters supporting GenPBS of t-bit precision to compute functions of t-bit input [49, 40] (while a negacyclic function of t-bit input can be computed by

[11] The default parameter of `tfhe-rs` library for `shortint` type supports GenPBS with 4-bit precision.

GenPBS of $(t-1)$-bit precision). Using the additional evaluation keys of large size requires an additional communication overload, so it might weaken the purpose of the transciphering framework. WoP-PBS of t-bit precision using GenPBS of t-bit precision is not compatible with the double blind rotation, a technique for the FRAST evaluation described in Section 4.2, since the double blind rotation requires one more padding bit to evaluate a non-negacyclic function. ComBo, proposed by Clet et al. [23], is the first WoP-PBS method that can compute an arbitrary function of t-bit precision using GenPBS of $(t-1)$-bit precision without additional evaluation keys. However, ComBo is also not compatible with the double blind rotation since it uses GenPBS operations in depth 2.

4.1 Our New WoP-PBS

In this section, we introduce our new WoP-PBS that can compute an arbitrary function of t-bit precision using at most 3 GenPBS operations of $(t-1)$-bit precision with no additional evaluation key. If it is possible to use the multi-value PBS [18] or PBSmanyLUT [22] together, our WoP-PBS can compute arbitrary functions in 2 GenPBS operations. In parallel computation, a variant of our WoP-PBS can fully parallelize the required GenPBS operations, obtaining the latency almost the same as that of a single GenPBS operation. Most importantly, our WoP-PBS method supports the double blind rotation technique, described in Section 4.2, to evaluate FRAST efficiently.

For an input message $m \in [\![0, p[\![$ with $m = \beta \| m' = \beta \cdot \frac{p}{2} + m'$ for $\beta \in \{0, 1\}$ and $m' \in [\![0, p/2[\![$, let $\mathsf{ClearMSB}(m) = 0 \| m'$, which can be computed in a single GenPBS operation by extracting the MSB of the input and subtracting it from the input on the MSB position. The resulting ciphertext can be regarded as a ciphertext of $\mathsf{ClearMSB}(m) \in [\![0, p/2[\![$ with one bit of padding.

The function f to compute is decomposed into $f_{\mathsf{msb\text{-}odd}}$ and $f_{\mathsf{msb\text{-}even}}$, which are defined as follows.

$$
\begin{aligned}
f_{\mathsf{msb\text{-}odd}}(m) &= \frac{1}{2}\left(f(m) - f\left(m + \frac{p}{2}\right)\right) = \frac{1}{2}\left(f(\beta \| m') - f(\overline{\beta} \| m')\right)\\
f_{\mathsf{msb\text{-}even}}(m) &= \frac{1}{2}\left(f(m) + f\left(m + \frac{p}{2}\right)\right) = \frac{1}{2}\left(f(\beta \| m') + f(\overline{\beta} \| m')\right)
\end{aligned}
$$

where $\overline{\beta} = 1 - \beta \in \{0, 1\}$, the flipped bit of β. From $f = f_{\mathsf{msb\text{-}odd}} + f_{\mathsf{msb\text{-}even}}$, one can compute f by computing $f_{\mathsf{msb\text{-}odd}}$ and $f_{\mathsf{msb\text{-}even}}$.

We note that $f_{\mathsf{msb\text{-}odd}}$ becomes a negacyclic function on $[\![0, p[\![$ from its definition, i.e., $f_{\mathsf{msb\text{-}odd}}(m + \frac{p}{2}) = -f_{\mathsf{msb\text{-}odd}}(m)$, so it can be evaluated by a single GenPBS operation. In the case of $f_{\mathsf{msb\text{-}even}}$, it can be regarded as a function on $[\![0, p/2[\![$ since $f_{\mathsf{msb\text{-}even}}(m + \frac{p}{2}) = f_{\mathsf{msb\text{-}even}}(m)$. Observing that the MSB of the input does not affect the output of $f_{\mathsf{msb\text{-}even}}$, one can compute $f_{\mathsf{msb\text{-}even}}$ in a single GenPBS operation with an input of the ciphertext of $\mathsf{ClearMSB}(m)$ with a padding bit. All the GenPBS operations are of $(\log p - 1)$-bit precision. Since the output ciphertext of our WoP-PBS is sum of two outputs of GenPBS operations, the error variance of it is two times larger than that of the GenPBS operation in the same parameters.

Our WoP-PBS requires 3 GenPBS operations in its naive evaluation: one for each of $f_{\mathsf{msb\text{-}odd}}$, $\mathsf{ClearMSB}$, and $f_{\mathsf{msb\text{-}even}}$. When the multi-value PBS [18] or PBSmanyLUT [22] can be used together, one can compute $f_{\mathsf{msb\text{-}odd}}$ and $\mathsf{ClearMSB}$ in a single GenPBS operation since they compute negacyclic functions on the same input.

Using parallel computation, a variant of our WoP-PBS can achieve almost the same latency with a single GenPBS operation as follows. After decomposing f into $f_{\mathsf{msb\text{-}odd}}$ and $f_{\mathsf{msb\text{-}even}}$, one can further decompose $f_{\mathsf{msb\text{-}even}}$ using the same method recursively by regarding $f_{\mathsf{msb\text{-}even}}$ as a function on $[\![0, p/2[\![$. For example, when $p = 16$, an arbitrary function f on $[\![0, p[\![$ is decomposed into 4 negacyclic functions f_0, f_1, f_2, f_3 and one constant f_4 such that

$$f(m) = f_0(m) + f_1(m \bmod 8) + f_2(m \bmod 4) + f_3(m \bmod 2) + f_4$$

where f_i is negacyclic on $[\![0, p/2^i[\![$ for $i = 0, \ldots, 3$. All the f_i can be computed by a single GenPBS of $(\log p - i - 1)$-bit precision without the padding bit. The input ciphertext of $m \bmod 2^{4-i}$ without padding can be obtained by using the plaintext modulus switching ignoring some of the first MSB bits [22], which is the same as multiplying 2^i to the original input. This decomposition can be generalized to the function on $[\![0, p[\![$ for arbitrary power-of-two p, resulting in $\log p$ negacyclic functions and one constant. All the decomposed functions can be computed in parallel, achieving the latency of a single GenPBS operation using $\log p$ threads. The error variance of the output ciphertext is $(\log p)^2$ times larger than the output of GenPBS in the worst-case[12]. For the exact specification of the decomposed functions, see Supplementary Material F.

4.2 Double Blind Rotation

The round function of FRAST requires multiple S-box calls on the inputs of the form $x_1 + rk_j$ for $j = 2, \ldots, \ell$ where x_1 is the value of the first branch, rk_j's are the round keys, and ℓ is the number of branches. In the GenPBS operation computing a function f on $x_1 + rk_j$, the blind rotation step rotates a GLWE ciphertext encoding f by a factor of $x_1 + rk_j$. In a naive way, evaluating $f(x_1 + rk_j)$ for all $j = 2, \ldots, \ell$ requires rotating a GLWE ciphertext encoding f by $x_1 + rk_j$ independently for all $j = 2, \ldots, \ell$.

The idea of the double blind rotation is that the result of the rotation by x_1 can be shared. Suppose an LWE ciphertext of x_1 without padding and GGSW ciphertexts of the round key bits $rk_{j,b}$ are given where $rk_j = rk_{j,4} \| \ldots \| rk_{j,1}$ for $j = 2, 3, \ldots, \ell$ and $b = 1, \ldots, 4$. For a negacyclic function f, let $\mathrm{GLWE}(P_f)$ be a GLWE ciphertext of $P_f \in \mathcal{R}_{q,N}$ that encodes f on its coefficients. The blind rotation on $\mathrm{GLWE}(P_f)$ by $\mathrm{LWE}(x_1)$ outputs $\mathrm{GLWE}(P_f \cdot X^{-\hat{x}_1})$ where $\hat{x}_1$ is a scaled value of x_1 such that the constant term of $P_f \cdot X^{-\hat{x}_1}$ becomes $f(x_1)$. Then

[12] Estimating the error variance as $\log p$ times larger one in the average-case requires the heuristic assumption that outputs of the GenPBS operations on the inputs that differ only by constant factors have independent errors.

one can compute GLWE$(P_f \cdot X^{-(\hat{x}_1 + \widehat{rk}_j)})$ by multiplying $X^{-\widehat{rk}_j}$ homomorphically where $\widehat{rk}_j$ is the scaled value of rk_j. Given the GGSW ciphertexts of the round key bits, it can be computed by additional 4 CMux gates. Therefore, one can compute LWE$(f(x_1 + rk_j))$ for all $j = 2, \ldots, \ell$ by a single GenPBS followed by $4(\ell - 1)$ CMux gates.

Two issues remain for applying the double blind rotation on FRAST. First, the fixed S-box in the FRAST round function is not negacyclic, requiring WoP-PBS for its evaluation instead of GenPBS. In this case, a WoP-PBS method that does not perform GenPBS operations in depth 2 is required. Evaluating a non-negacyclic function with the help of padding bits is also possible, but the double blind rotation requires two bits of padding to guarantee $\hat{x_1} + \widehat{rk}_i < N$. Our WoP-PBS method can resolve this issue for evaluating FRAST with TFHE parameters supporting a GenPBS operation in 4-bit precision. When S is decomposed into $S_{\mathsf{msb\text{-}odd}}$ and $S_{\mathsf{msb\text{-}even}}$, the double blind rotation can be applied to $S_{\mathsf{msb\text{-}odd}}$ as it is negacyclic. In the case of $S_{\mathsf{msb\text{-}even}}$, one can make two bits of padding in the ciphertext of $\mathsf{ClearMSB}(x_1)$ by one more GenPBS operation to change its scaling factor, allowing the double blind rotation using a GenPBS operation of 4-bit precision[13]. See Supplementary Material H for details of FRAST evaluation by the double blind rotation.

The other is computing the GGSW ciphertexts of the round key bits. Since the round key bits are fixed, we directly transfer the round key bits used for the double blind rotation packed in the GLWE ciphertexts once, and convert it into GGSW ciphertexts on the server-side by the GLWEtoGGSW conversion proposed in [19][14]. The communication overload for the round keys and the evaluation keys for the conversion is only a few MBs.

5 Security Analysis

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. We also limit the number of encryptions under the same key up to 2^{64} blocks since otherwise one would not be able to avoid a nonce collision (when nonces are chosen uniformly at random).

The extendable output function whose output determines the random S-boxes is modeled as a random oracle, so an adversary is not able to freely choose the S-boxes. The input to the FRAST is also fixed as the known constant **ic**. Therefore, in this model, we believe that FRAST is secure against any type of chosen-plaintext attacks such as (higher-order) differential, truncated differen-

[13] Using the variant of our WoP-PBS that fully decomposes a function into negacyclic functions enables the double blind rotation using a GenPBS operation of 3-bit precision at the cost of more CMux gate operations.

[14] It only deals with the case of converting RLWE to RGSW, but converting GLWE to GGSW for $k > 1$ is also possible using the same idea.

tial, invariant subspace trail, and cube attacks. On the other hand, we assume that the specifications of the random S-boxes are given to the attacker.

Overall, in this section, our focus will be mainly put on algebraic and linear attacks, which are possible in the known-plaintext models. We analyze the security of FRAST against algebraic (resp. linear) attacks based on the fixed (resp. random) round functions of FRAST.

5.1 Algebraic Attacks

An algebraic attack is to build a system of polynomial equations and solve it to recover the secret key. For simplicity, we ignore the random rounds of FRAST. Instead, we assume that a distinct input is given to FRAST since the first 4 rounds are randomized ones. This can be considered as a case such that all the random S-boxes are linear, which is in favor of an attacker, so that the algebraic degree does not increase in the random rounds.

Algebraic Representation over $\mathbb{Z}_{16}$. Recently, Grassi et. al. [35] showed that the brute-force attack on a polynomial system of n secret elements over $\mathbb{Z}_q$, where $q = p^y$ for a prime p and $y > 1$, requires only $O(y \cdot p^n)$ computation instead of $O(p^{y \cdot n})$. This implies that if FRAST has a polynomial representation over $\mathbb{Z}_{16}$ then it becomes vulnerable to the brute-force search attack. On the other hand, for an S-box to have a polynomial representation over $\mathbb{Z}_{16}$, the following condition should be satisfied.

Lemma 1. *Let $f : \mathbb{Z}_{16} \to \mathbb{Z}_{16}$ be a function over $\mathbb{Z}_{16}$. If f has a polynomial representation, then it should satisfy $f(i+8)-f(i) \in \{0,8\}$ for all $i = 0,1,\ldots,7$.*

Proof. Suppose that f is represented by $f(x) = a_0 + a_1x + \cdots + a_{15}x^{15}$ where $a_0, a_1, \ldots, a_{15} \in \mathbb{Z}_{16}$. Then, for all $i = 0,1,\ldots,7$, one obtains

$$f(i+8) - f(i) = 8\left(\sum_{j=1}^{15} a_j \left(\sum_{m=0}^{j-1} (i+8)^{j-m-1} i^m\right)\right).$$

Since $8z \in \{0,8\}$ for every $z \in \mathbb{Z}_{16}$, $f(i+8) - f(i) \in \{0,8\}$. □

As the fixed S-box of FRAST does not satisfy the necessary condition above, we can conclude that the FRAST round function cannot be represented as a polynomial over $\mathbb{Z}_{16}$.

Algebraic Analysis on the XOR Variant of FRAST. We consider an XOR-variant of FRAST with addition and constant multiplication over $\mathbb{Z}_{16}$ replaced by XOR and multiplication over $\mathbb{F}_{2^4}$. Such an approach has also been taken for the algebraic analysis of the Elisabeth cipher by introducing an XOR-variant of Elisabeth, dubbed Beth.

The XOR-variant of FRAST can be represented by a system of Boolean equations. In this section, we analyze FRAST using various systems of Boolean equations for its XOR-variant. For the complexity of linearization attacks, we will use $\omega = 2$ in (2) and (3). In particular, for the XL attack, the independence assumption will be used to estimate the solving degree as mentioned in Section 2.3.1.

5.1.1 Trivial Linearization

One can build a system of equations using only the key variables as unknowns and apply trivial linearization attack. The attack cost depends on the number of monomials appearing in the system, determined by the degree of the system.

Consider a single round function of FRAST with input $(x_1, x_2, \ldots, x_\ell)$, output $(y_1, y_2, \ldots, y_\ell)$, and round key $(rk_1, rk_2, \ldots, rk_\ell)$ where $x_j, y_j, rk_j \in \mathbb{F}_2^4$ for $j = 1, 2, \ldots, \ell$[15]. Then we have

$$\begin{aligned} y_1 &= x_1 + S(y_2 + y_3 + \cdots + y_\ell + rk_1), \\ y_j &= x_j + S(x_1 + rk_j) && \text{for } j = 2, 3, \ldots, \ell. \end{aligned} \tag{6}$$

Since all the outputs of S are of degree 3, the degree of $S(x + rk)$ is at least $\deg x + 2$, assuming that rk_j is a dense linear combination of the master key. Then, we have $\deg y_j \geq \deg x_1 + 2$ for $j = 2, \ldots, \ell$ and $\deg y_1 \geq \deg x_1 + 4$. Let $(x_1^{(i-1)}, \ldots, x_\ell^{(i-1)})$ (resp. $(x_1^{(i)}, \ldots, x_\ell^{(i)})$) be the input (resp. output) of the i-th round function. From $\deg x_j^{(1)} = 3$ for $j = 2, \ldots, \ell$, we obtain

$$\begin{aligned} \deg x_1^{(i)} &\geq 4i + 1, \\ \deg x_j^{(i)} &\geq 4i - 1 && \text{for } j = 2, \ldots, \ell \end{aligned}$$

for $i = 1, 2, \ldots, r$ where r is the number of rounds.

Considering the meet-in-the-middle attack, we also need to consider the backward direction. Using the same argument, one can obtain a system of degree at least $4\lfloor r/2 \rfloor + 1$, which is 17 when $r = 8$.

Since the round keys are dense linear combinations of the master key, one can expect almost all the monomials of degree up to the lower bound to appear in the system. Experimental results on the number of the appearing monomials with toy parameters are given in Supplementary Material C. Assuming the system of FRAST contains almost all the monomials of degree 17, the time complexity of the (hybrid) trivial linearization attack is $2^{173.76}$.

Remark 1. The attack on Elisabeth [34] is one of the trivial linearization attacks combined with an optimization technique on the Gaussian elimination. Hence, FRAST is also secure against the attack.

5.1.2 XL Attack

Other than the equations only in the key variables, one can build a system of equations of low degrees by introducing new variables other than the key variables or guessing some bits of the internal state. Then it is possible to apply algebraic attacks such as the XL attack and the Gröbner basis attack. We consider the following three kinds of systems for FRAST.

1. A system of equations by introducing new intermediate variables for each round.

[15] $\ell = 32$ in the actual specification of FRAST.

2. A system of equations by introducing new variables for the first branch of each round.
3. A system of equations by guessing the values of the first branch for each round.

New Variables for the Intermediate State. One can build a system of equations by introducing new intermediate variables for each round. In this case, using implicit relations induced by S leads to a larger number of equations of lower degrees than using explicit relations. Let $(x_j^{(i-1)})_{j=1}^{\ell}$ and $(x_j^{(i)})_{j=1}^{\ell}$ denote the input and the output of the i-th round function, respectively. From the implicit relation $S'(x,y)=0$ induced by S, one can obtain the following equations.

$$\begin{aligned} &S'\big(x_2^{(i)}+x_3^{(i)}+\cdots+x_\ell^{(i)}+rk_1^{(i)}, x_1^{(i)}-x_1^{(i-1)}\big)=0 \\ &\qquad\qquad\qquad S'\big(x_1^{(i-1)}+rk_j^{(i)}, x_j^{(i)}-x_j^{(i-1)}\big)=0 \quad \text{for } j=2,3,\dots,\ell \end{aligned} \tag{7}$$

where $(rk_j^{(i)})_{j=1}^{\ell}$ denotes the i-th round key which is linear to the master key.

Since S-box S has 21 implicit quadratic equations over $\mathbb{F}_2$, each round induces 21ℓ quadratic equations. For r rounds of FRAST, one obtains $21\ell r$ quadratic equations in $256+128(r-1)$ variables. When m keystream blocks are used, one obtains $21\ell rm$ quadratic equations in $256+128(r-1)m$ variables.

New Variables for the First Branch. Since the first branch is common to all the S-box evaluations, introducing new variables for $(x_1^{(i)})_{i=1}^{r-1}$ might significantly reduce the degree of the keystream $x_j^{(r)}$ for all $j=2,3,\dots,\ell$. Regarding $(x_1^{(i)})_{i=1}^{r-1}$ as new variables, one obtains two types of equations: one from the keystream $(x_j^{(r)})_{j=2}^{\ell}$ and the other from $(x_1^{(i)})_{i=1}^{r}$. The first type of equations are

$$x_j^{(r)} = x_j^{(0)} + \sum_{i=1}^{r} S(x_1^{(i-1)} + rk_j^{(i)}) \tag{8}$$

for $j=2,3,\dots,\ell$, and the other type of equations are

$$rk_1^{(i)} + \sum_{j=2}^{\ell} x_j^{(i)} = S^{-1}(x_1^{(i)} - x_1^{(i-1)}) \tag{9}$$

where $x_j^{(i)} = x_j^{(0)} + \sum_{t=1}^{i} S(x_1^{(t-1)} + rk_j^{(t)})$ for $i=1,2,\dots,r$. Hence, we obtain $4(\ell+r-1)$ equations of degree 3 in $256+4(r-1)$ variables. From m keystream blocks, one obtains $4(\ell+r-1)m$ equations of degree 3 in $256+4(r-1)m$ variables.

Remark 2. Both equations (8) and (9) use the explicit representation of S. If the implicit representation is used, then the resulting system of equations is of degree 4. Since the XL attacks are based on the strategy of extending equations of lower degrees to a larger number of equations of higher degrees by multiplying polynomials (or monomials), we only consider the system of equations of a lower degree if the same variables are used.

Guessing the First Branch. It is possible to guess the intermediate states of the first branch $x_1^{(i)}$ for $i = 1, 2, \ldots, r-1$, in which case we obtain equations of the same degree except that equation (9) becomes linear for $i = r$ since $x_j^{(r)}$ are known for $j = 1, 2, \ldots, \ell$. This linear equation determines the value of $rk_1^{(r)}$, reducing the number of variables by 4. Hence, by guessing $4(r-1)$ bits of the intermediate state, one obtains a system of $4(\ell + r - 2)$ equations of degree 3 in 252 variables.

When m keystream blocks are used, the round key $rk_1^{(1)}$ fixed in the first block determines $x_1^{(r-1)}$ for all the other blocks. Therefore, one obtains a system of $4(\ell + r - 2)m$ equations of degree 3 in 252 variables by guessing $4(r-1) + 4(r-2)(m-1)$ bits of the intermediate state of the first branch.

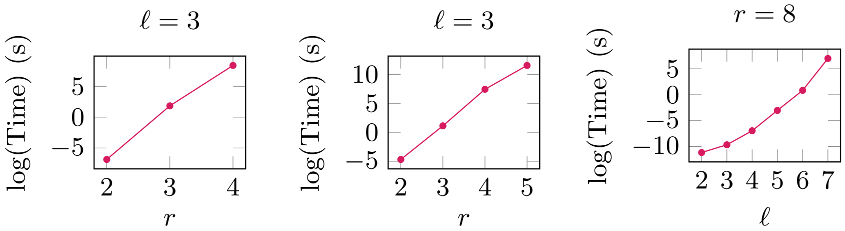
XL Attack Complexity. Table 2 summarizes the complexity of the XL attacks for the above systems according to the number of keystream blocks used to build the systems. One can find that FRAST is secure against the XL attacks under 128-bit security even with the independent assumption.

# Blocks	Intermediate Variables					First Branch Variables					Guessing First Branch				
	# Var	# Eqs	g_{opt}	D	Cost	# Var	# Eqs	g_{opt}	D	Cost	# Var	# Eqs	g_{opt}	D	Cost
1	1152	5376	3	16	239.56	284	156	127	26	323.85	252	152	97	26	320.82
2	2045	10752	10	20	327.32	312	312	120	26	333.02	252	304	70	25	335.01
3	2944	16128	0	24	394.81	340	468	124	26	346.39	252	456	46	25	335.01
4	3840	21504	0	27	456.45	368	624	8	39	356.90	252	760	18	26	359.92

Table 2: Complexity of the hybrid XL attacks for each system. g_{opt} denotes the optimal number of guessing for the hybrid XL attack, D denotes the solving degree, and 'Cost' denotes the attack complexity in bits. For the system guessing the first branch, guessing $32m$ bits of the first branch is considered in its cost where m is the number of blocks.

5.1.3 Gröbner Basis Attack

We experimentally computed Gröbner basis for the systems described in Section 5.1.2 on toy parameters and found that the highest degree reached during the computation is not well estimated by the degree of regularity from (4). Instead, we experimentally verified that the actual Gröbner basis computation time grows exponentially according to the number of rounds and the number of branches. Figure 4 shows the Gröbner basis computation time of the systems obtained from a single input-output pair. For the systems obtained from multiple input-output pairs, see Supplementary Material D.



(a) Intermediate Variables (b) First Branch Variables (c) Guess First Branches

Fig. 4: Gröbner basis computation time of the systems on toy parameters. The number of rounds and branches are denoted by r and ℓ, respectively. The key size is set to 4ℓ bits, which is half the actual key size.

5.1.4 Other Algebraic Attacks

The fast exhaustive search attack [14] can be seen as an optimized variant of the brute-force search. It evaluates Boolean equations of degree d in n variables using $d \cdot 2^n$ bit operations. However, the key length of 256 bits mitigates the fast exhaustive search attack against FRAST.

The polynomial method [7] is a technique for solving a system of multivariate polynomial equations. For a system of Boolean equations of degree d in n variables, its time complexity is estimated as $n^2 \cdot 2^{(1-1/(2.7d))n}$ in bit operations [28]. The polynomial method is also infeasible for FRAST using a secret key of 256 bits.

The interpolation attack [38] is to establish an encryption polynomial only in plaintext variables by considering the secret key as an (unknown) constant. The input to FRAST is fixed as constant **ic** and the encryption function is distinct for every encryption, so it seems infeasible to apply the interpolation attack in a straightforward manner. On the other hand, one might try to establish a polynomial in the XOF outputs (that determine the random S-boxes). In this case, however, the number of variables is much larger and the degree growth by lookup operation on the random S-box is much faster compared to the systems only in key variables described in Section 5.1.1.

5.2 Linear Attacks

The linear attack was originally introduced for binary spaces [41], but it can also be applied to non-binary spaces [3]. The linear probability of a function $\mathsf{E} : \mathbb{Z}_p^\ell \to \mathbb{Z}_p^\ell$ with input and output masks $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^\ell$ is defined as follows:

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

where $\mathbf{x}$ is uniformly distributed over $\mathbb{Z}_p^\ell$. We refer to [3] for the details.

Traditional linear cryptanalysis that requires many input-output pairs of a fixed function does not apply to FRAST since the keystream generating function of FRAST changes for each keystream block. So in order to analyze the resistance of FRAST against linear cryptanalysis, we consider the following three strategies to be combined with linear attacks.

1. Applying nonzero linear masks on the XOF outputs generating the random S-boxes of FRAST.
2. Collecting input-output pairs that can be analyzed by the same linear trail.
3. Collecting input-output pairs whose random S-boxes have linear relations.

In this section, we show how the above strategies are mitigated by the randomly generated S-boxes of FRAST.

Nonzero Linear Masks on the XOF Outputs. The first approach is to apply nonzero linear masks on the XOF outputs that determine the random S-boxes of FRAST. Although the XOF outputs themselves are not controllable by an attacker, they can be considered as additional inputs in the KPA model since they are publicly known. Using this approach, one can apply the linear cryptanalysis to FRAST by considering it as a fixed function whose input size is larger than its output size. For example, a negacyclic random S-box S itself can be described by its function values $(S(0), \dots, S(7))$, and the function $\mathsf{LookUp}_k(x, S) = S(x + k)$ can be described as a function from $\mathbb{Z}_{16} \times \mathbb{Z}_{16}^8$ to $\mathbb{Z}_{16}$ defined by

$$\mathsf{LookUp}_k(x, S) = \sum_{i=0}^{7} \left(\mathbf{1}\left\{x + k = i\right\} - \mathbf{1}\left\{x + k = i + 8\right\} \right) S(i)$$

where $\mathbf{1}\left\{x + k = i\right\} = 1$ if $x+k = i$ and 0 otherwise. From this point of view, the linear probability of FRAST with additional linear masks on the XOF outputs is well-defined.

However, unlike the traditional linear cryptanalysis model, the resulting linear probability depends on the relation between the linear masks (on the XOF outputs) and the round keys. Hence, the masks should be chosen carefully considering the relation with the round keys. Otherwise, the linear probability becomes zero.

One possible choice is to use a trail with nonzero linear probability regardless of the round keys. We find it possible to build such a trail when only one S-box is activated for each round, and its linear probability is 2^{-6} for each round. As FRAST has 32 rounds of random S-boxes, the linear probability of the trail becomes 2^{-192}. If more than two S-boxes are activated, then the linear masks and the round key should satisfy a certain relation. To build a linear trail satisfying the relation, the attacker should guess at least one round key (or a difference between round keys). Since there are 32 rounds of random S-boxes, it requires guessing at least 128 bits of the round keys. We refer to Supplementary Material E.1 for the details.

Using Compatible Input-Output Pairs. The next approach is to apply the traditional linear cryptanalysis on data that can be analyzed under the same linear trail. Suppose one finds a good linear trail on a single input/output pair of FRAST. In general, applying the trail on the other input/output pair does not work since the randomly generated S-boxes are different for the two data. However, if the S-boxes of two data work identically with respect to the linear approximation by the trail, one can analyze two data using the trail.

To consider such a case, we define two S-boxes S_1 and S_2 are *compatible* with respect to the output linear mask b if $b(S_1 - S_2)$ is constant[16]. The probability that two randomly chosen negacyclic S-boxes are compatible is at most 2^{-7}. Since there is at least one active S-box in the non-trivial linear trail of FRAST, the probability that two data are compatible with respect to a linear trail is at most 2^{-224}, which is negligible under 128-bit security. We refer to Supplementary Material E.2 for the details.

Using Linear Relations of Random S-boxes. One might use data from multiple instances of FRAST whose random S-boxes have certain linear relations. If the S-box has a linear relation such as $ax + bS(x) = c$ for all x, then one can construct input/output linear masks of linear probability 1 for the corresponding round. When all the active S-boxes have linear relation, then one can use a linear trail whose linear probability depends only on the fixed rounds. Since the probability that a random negacyclic S-box has a linear relation is at most 2^{-6} (see Supplementary Material E.3 for the details), the probability that all the active random S-boxes have such linear relations is at most 2^{-192}, which is sufficiently small at the security level of 128 bits.

Zero-Correlation Attack. In contrast to the classical linear attack finding a high linear correlation, Bogdanov and Rijmen [9] proposed a variant of linear attack using linear hulls with correlation zero. This attack is based on the assumption that there might be a linear hull of correlation zero for every secret key (due to a certain specific structure of the block cipher), while it is not the case for a truly random permutation. Hence if one knows such a linear hull of correlation zero and collects 2^{n-1} input-output pairs under the same key, then the block cipher can be distinguished from a random permutation. Later, Bogdanov and Wang [10] reduced the data complexity down to $O(2^n/\sqrt{\ell})$, where ℓ is the number of the linear hulls with correlation zero.

The zero-correlation attack is not applicable to FRAST since the output keystream blocks of FRAST are not produced by a fixed permutation. To generate keystream blocks of FRAST, we feed a fixed input **ic** to different encryption functions based on random S-boxes. If each keystream block of FRAST is regarded as an output of an independent permutation, then there will be no correlation between the keystream blocks, giving no distinguishing advantage to an adversary.

[16] $b(S_1 - S_2)$ need not to be zero since the constant difference does not affect the linear probability.

6 Performance Evaluation

In this section, we evaluate the server-side performance of FRAST in the transciphering framework with TFHE and compare it with Elisabeth (and their patches) and Kreyvium. We omit the previous result for FiLIP proposed by Hoffmann et al. [36] since it takes more than a second to evaluate a single bit, which is far slower than Elisabeth and Kreyvium. Méaux et al. [44] proposed a FINAL-based FiLIP evaluation method with a notable performance in terms of computation time, while we believe that it is hard to make an apples-to-apples comparison of the benchmark since it requires much larger key size of about 1 GB[17] (even might be larger in the TFHE scheme). For the patches of Elisabeth, we only consider Elisabeth-b and Gabriel since Margrethe requires to evaluate a lookup table of 18-bit inputs, which is impractical[18].

The source codes of the server-side computation are developed in Rust with `tfhe-rs` library [50] which supports the TFHE scheme. The extendable output function XOF has been instantiated with SHAKE256. Our experiments are executed in Intel i5-13600K @ 3.90 GHz[19].

For homomorphic evaluation of FRAST, the default parameters of `tfhe-rs` library for `shortint` type, named `PARAM_MESSAGE_2_CARRY_2`, are used along with some chosen parameters for the GLWEtoGGSW conversion. Specifically, the following parameters have been used.

- GenPBS parameter
  - LWE parameters: $n = 742$, $\sigma_{\text{LWE}} = 7.06984 \times 10^{-6}$
  - GLWE parameters: $k = 1$, $N = 2048$, $\sigma_{\text{GLWE}} = 2.94036 \times 10^{-16}$
  - PBS parameters: $\log B_{\mathsf{PBS}} = 23$, $\ell_{\mathsf{PBS}} = 1$
  - Keyswitching parameters: $\log B_{\mathsf{KS}} = 3$, $\ell_{\mathsf{KS}} = 5$
- GLWEtoGGSW parameter
  - GGSW parameters of the GLWE secret key: $\log B_{\mathsf{SK}} = 9$, $\ell_{\mathsf{SK}} = 5$
  - GLWE keyswitching parameters: $\log B_{\mathsf{subs}} = 9$, $\ell_{\mathsf{subs}} = 5$
  - GGSW parameters of the round key bits: $\log B_{\mathsf{rk}} = 7$, $\ell_{\mathsf{rk}} = 3$

With the above parameters, 128-bit security is achieved and the error probability is upper bounded by 2^{-40}. See Supplementary Material H for detailed error analysis. For Elisabeth and Kreyvium, the TwoKS parameters in [25] and the parameters in [4] are used, respectively. For the performance evaluation, we consider the case where the actual parameters after the transciphering are the default parameters of `tfhe-rs` library for `shortint` type as in [4].

[17] The `PARAM_MESSAGE_2_CARRY_2` parameters correspond to the case of $p = 2^5$ with Set-II.

[18] The authors also left the homomorphic evaluation of Margrethe as an open problem.

[19] It has 6 P-cores @ 5.30 GHz and 8 E-cores @ 3.90 GHz, and we only used the 8 E-cores for the benchmark.

6.1 Benchmark and Comparison

The transciphering framework with a stream cipher requires only simple subtraction in the online phase, while, when it comes to TFHE, additional computation is required after the subtraction for plaintext encoding such as clearing carry bits and matching the plaintext encoding. Kreyvium clears the carry bit after subtraction by a single KSthenGenPBS operation [4]. When it comes to Elisabeth producing ciphertexts of 4-bit keystream blocks without padding, Cosseron et al. proposed to use one-bit smaller plaintext space with padding for the resulting ciphertext [25]. Although the online phase only requires homomorphic subtraction for this plaintext encoding, it is not commonly used in TFHE. Most importantly, the ciphertext expansion ratio becomes greater than 1 since each ciphertext of 4 bits only contains a plaintext of 3 bits.

FRAST produces ciphertexts of 4-bit keystream blocks without padding, too. In order to support various types of plaintext encoding without ciphertext expansion, we add a bit extraction process at the end of the offline phase; a ciphertext of 4 bits is decomposed into 4 ciphertexts of a single bit whose plaintext encoding supports 1-bit plaintext space without carry and padding space. This plaintext encoding is also called 2-encoding in [11]. This extraction can be done homomorphically at the cost of almost one PBS by the multi-value PBS [18] since the extraction functions are negacyclic under the 2-encoding.

The online phase computes homomorphic subtraction, a single PBS operation to match the plaintext encoding of the resulting ciphertext, and a keyswitching to the final TFHE parameters. We note that the online phase performance does not depend on the cipher, but on the TFHE parameters it uses. Hence, by using a faster bootstrapping key for the online phase, the online performance can be improved. For FRAST, we use the bootstrapping key of Kreyvium for the bit extraction and the online phase. The bootstrapping key of Elisabeth has a similar performance to that of Kreyvium, so we can say that all the ciphers have almost the same online performance.

Cipher	Setup (s)	Keystream Evaluation	
		Lat. (ms)	Thrp. (bit/s)
Elisabeth	-	2049 (per 4-bit)	1.955
Elisabeth-b	-	5538 (per 4-bit)	0.749
Gabriel	-	4662 (per 4-bit)	0.858
Kreyvium	44.09 (4 threads)	134.0 (per 1-bit)	7.465
FRAST	24.99 (8 threads)	6194 (per 128-bit)	20.66

Table 3: Server-side offline phase performance. The setup time of Kreyvium is optimized in 4 threads. Keystream evaluation is performed in a single thread.

Offline Phase. The offline phase consists of the setup phase and the keystream evaluation phase. The setup phase is performed only once so that the latency (optimized by multithreading) should be seen as a more appropriate metric than the throughput. The setup time of Kreyvium is estimated as 1152 cycles for the main loop, where each cycle is estimated by 2 KSthenGenPBS operations using 4 threads[20]. The setup time of FRAST is spent to convert the GLWE ciphertexts packing round key bits into GGSW ciphertexts of each round key bit, which can be optimized by using multiple threads. On the other hand, Elisabeth and its patches have no setup phase.

The latency for the keystream evaluation can be reduced by using multiple threads, while it is more efficient to evaluate each keystream block independently by each thread in terms of throughput. The offline performance of Kreyvium is estimated by only 7 KSthenGenPBS operations, evaluating a single keystream bit. The offline performance of Elisabeth is estimated by 96 KSthenGenPBS operations, evaluating a 4-bit keystream block, followed by 1 KSthenGenPBS operations for bit extraction. In case of Elisabeth-b (resp. Gabriel), evaluating 4-bit keystream block requires 252 (resp. 220) KSthenGenPBS operations, and the bit extraction requires 1 KSthenGenPBS. The result is summarized in Table 3. One can see that FRAST outperforms Elisabeth and Kreyvium in terms of throughput by factors of 10.57 and 2.768, respectively.

Online Phase. For all the ciphers, the online performance is estimated by two GenPBS operations followed by a single keyswitching to the default parameters for a ciphertext of a 2-bit message, and it only depends on the online phase parameters. In our setting, we obtain the latency of 46.08 ms for a ciphertext of a 2-bit message, which implies the throughput of 43.40 bit/s.

Communication Overload. Communication overload is mainly due to homomorphic ciphertexts of the secret keys and the TFHE evaluation keys for the transciphering. The communication overload of Kreyvium (resp. Elisabeth) is estimated as 10.72 MB (resp. 12.29 MB).

When it comes to FRAST, the bootstrapping keys for the default parameters can be recycled in the actual usecase after transciphering, reducing the overall communication overload. Instead, the bootstrapping keys of Kreyvium are used for the bit extraction and the online phase, and additional evaluation keys for the double blind rotation are required. In this way, the communication overload for FRAST is estimated as 11.88 MB. See Supplementary Material G for details.

References

[1] Albrecht, M.R., Grassi, L., Perrin, L., Ramacher, S., Rechberger, C., Rotaru, D., Roy, A., Schofnegger, M.: Feistel Structures for MPC, and More.

[20] The setup time for Kreyvium in [4] (which is called warm-up time) is obtained by dividing the time for 1152 cycles by 64, considering the bit size of `FheUint64` type it computes. In this paper, we consider the total time required for initialization.

In: Sako, K., Schneider, S., Ryan, P.Y.A. (eds.) Computer Security – ESORICS 2019. pp. 151–171. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-29962-0_8

[2] Ars, G., Faugère, J.C., Imai, H., Kawazoe, M., Sugita, M.: Comparison Between XL and Gröbner Basis Algorithms. In: Lee, P.J. (ed.) ASIACRYPT 2004. pp. 338–353. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30539-2_24

[3] 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). https://doi.org/10.1007/978-3-540-77360-3_13

[4] Balenbois, T., Orfila, J.B., Smart, N.P.: Trivial Transciphering With Trivium and TFHE. Cryptology ePrint Archive, Paper 2023/980 (2023), https://eprint.iacr.org/2023/980, to appear WAHC 2023

[5] Bardet, M., Faugere, J.C., Salvy, B.: On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equations. In: Proceedings of the International Conference on Polynomial System Solving. pp. 71–74 (2004)

[6] Bardet, M., Faugère, J.C., Salvy, B., Spaenlehauer, P.J.: On the complexity of solving quadratic Boolean systems. Journal of Complexity **29**(1), 53–75 (2013). https://doi.org/10.1016/j.jco.2012.07.001

[7] Beigel, R.: The polynomial method in circuit complexity. In: [1993] Proceedings of the Eigth Annual Structure in Complexity Theory Conference. pp. 82–95 (1993). https://doi.org/10.1109/SCT.1993.336538

[8] Bergerat, L., Boudi, A., Bourgerie, Q., Chillotti, I., Ligier, D., Orfila, J.B., Tap, S.: Parameter Optimization and Larger Precision for (T)FHE. Journal of Cryptology **36**, 28 (2023). https://doi.org/10.1007/s00145-023-09463-5

[9] Bogdanov, A., Rijmen, V.: Linear hulls with correlation zero and linear cryptanalysis of block ciphers. Designs, codes and cryptography **70**, 369–383 (2014). https://doi.org/10.1007/s10623-012-9697-z

[10] Bogdanov, A., Wang, M.: Zero Correlation Linear Cryptanalysis with Reduced Data Complexity. In: FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers. pp. 29–48. Springer (2012). https://doi.org/10.1007/978-3-642-34047-5_3

[11] Bon, N., Pointcheval, D., Rivain, M.: Optimized Homomorphic Evaluation of Boolean Functions. Cryptology ePrint Archive, Paper 2023/1589 (2023), https://eprint.iacr.org/2023/1589

[12] Bonte, C., Iliashenko, I., Park, J., Pereira, H.V.L., Smart, N.P.: FINAL: Faster FHE Instantiated with NTRU and LWE. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022. pp. 188–215. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-22966-4_7

[13] Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system. I. The user language. J. Symbolic Comput. **24**(3-4), 235–265 (1997). https://doi.org/10.1006/jsco.1996.0125, http://dx.doi.org/10.1006/jsco.1996.0125, computational algebra and number theory (London, 1993)

[14] Bouillaguet, C., Chen, H.C., Cheng, C.M., Chou, T., Niederhagen, R., Shamir, A., Yang, B.Y.: Fast Exhaustive Search for Polynomial Systems in $\mathbb{F}_2$. In: CHES 2010. pp. 203–218. Springer (2010). https://doi.org/10.1007/978-3-642-15031-9_14

[15] Brakerski, Z.: Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. vol. 7417, pp. 868–886. Springer (2012). https://doi.org/10.1007/978-3-642-32009-5_50

[16] 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). https://doi.org/10.1145/2633600

[17] 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). https://doi.org/10.1007/s00145-017-9273-9

[18] Carpov, S., Izabachène, M., Mollimard, V.: New Techniques for Multi-value Input Homomorphic Evaluation and Applications. In: Matsui, M. (ed.) CT-RSA 2019. pp. 106–126. Springer (2019). https://doi.org/10.1007/978-3-030-12612-4_6

[19] Chen, H., Chillotti, I., Ren, L.: Onion Ring ORAM: Efficient Constant Bandwidth Oblivious RAM from (Leveled) TFHE. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. p. 345–360. CCS '19, ACM (2019). https://doi.org/10.1145/3319535.3354226

[20] Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast Fully Homomorphic Encryption Over the Torus. Journal of Cryptology **33**, 34–91 (2020). https://doi.org/10.1007/s00145-019-09319-x

[21] Chillotti, I., Joye, M., Paillier, P.: Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks. In: Dolev, S., Margalit, O., Pinkas, B., Schwarzmann, A. (eds.) Cyber Security Cryptography and Machine Learning. pp. 1–19. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-78086-9_1

[22] Chillotti, I., Ligier, D., Orfila, J.B., Tap, S.: Improved Programmable Bootstrapping with Larger Precision and Efficient Arithmetic Circuits for TFHE. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021. pp. 670–699. Springer (2021). https://doi.org/10.1007/978-3-030-92078-4_23

[23] Clet, P.E., Boudguiga, A., Sirdey, R., Zuber, M.: ComBo: A Novel Functional Bootstrapping Method for Efficient Evaluation of Nonlinear Functions in the Encrypted Domain. In: El Mrabet, N., De Feo, L., Duquesne, S. (eds.) AFRICACRYPT 2023. pp. 317–343. Springer (2023). https://doi.org/10.1007/978-3-031-37679-5_14

[24] Cong, K., Das, D., Park, J., Pereira, H.V.: SortingHat: Efficient Private Decision Tree Evaluation via Homomorphic Encryption and Transciphering. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. p. 563–577. CCS '22, ACM (2022). https://doi.org/10.1145/3548606.3560702

[25] Cosseron, O., Hoffmann, C., Méaux, P., Standaert, F.X.: Towards Case-Optimized Hybrid Homomorphic Encryption. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022. pp. 32–67. Springer (2022). https://doi.org/10.1007/978-3-031-22969-5_2

[26] Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 392–407. Springer (2000). https://doi.org/10.1007/3-540-45539-6_27

[27] De Cannière, C., Preneel, B.: Trivium. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs: The eSTREAM Finalists. pp. 244–266. Springer (2008). https://doi.org/10.1007/978-3-540-68351-3

[28] Dinur, I.: Cryptanalytic Applications of the Polynomial Method for Solving Multivariate Equation Systems over GF(2). In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021. pp. 374–403. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77870-5_14

[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] Faugere, J.C.: A new efficient algorithm for computing Gröbner bases (F_4). Journal of Pure and Applied Algebra **139**(1-3), 61–88 (1999). https://doi.org/10.1016/S0022-4049(99)00005-5

[31] Faugère, J.C.: A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In: Proceedings of the 2002 international symposium on Symbolic and algebraic computation. pp. 75–83. ACM (2002). https://doi.org/10.1145/780506.780516

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

[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.) CRYPTO 2013. vol. 8042, pp. 75–92. Springer (2013). https://doi.org/10.1007/978-3-642-40041-4_5

[34] Gilbert, H., Boissier, R.H., Jean, J., Reinhard, J.R.: Cryptanalysis of Elisabeth-4. Cryptology ePrint Archive, Paper 2023/1436 (2023), https://eprint.iacr.org/2023/1436, to appear ASIACRYPT 2023

[35] Grassi, L., Manterola Ayala, I., Hovd, M.N., Øygarden, M., Raddum, H., Wang, Q.: Cryptanalysis of Symmetric Primitives over Rings and a Key Recovery Attack on Rubato. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023. pp. 305–339. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-38548-3_11

[36] 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.) INDOCRYPT 2020. pp. 39–61. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-65277-7_3

[37] Hoffmann, C., Méaux, P., Standaert, F.X.: The Patching Landscape of Elisabeth-4 and the Mixed Filter Permutator Paradigm. Cryptology ePrint Archive, Paper 2023/1895 (2023), https://eprint.iacr.org/2023/1895, https://eprint.iacr.org/2023/1895

[38] Jakobsen, T., Knudsen, L.R.: The Interpolation Attack on Block Ciphers. In: FSE'97. pp. 28–40. Springer (1997). https://doi.org/10.1007/BFb0052332

[39] Kluczniak, K., Schild, L.: FDFB: Full Domain Functional Bootstrapping Towards Practical Fully Homomorphic Encryption. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(1), 501–537 (Nov 2022). https://doi.org/10.46586/tches.v2023.i1.501-537

[40] Liu, Z., Micciancio, D., Polyakov, Y.: Large-Precision Homomorphic Sign Evaluation Using FHEW/TFHE Bootstrapping. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022. pp. 130–160. Springer (2022). https://doi.org/10.1007/978-3-031-22966-4_5

[41] Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT '93. vol. 765, pp. 386–397. Springer (1994). https://doi.org/10.1007/3-540-48285-7_33

[42] 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.) INDOCRYPT 2019. vol. 11898, pp. 68–91. Springer (2019). https://doi.org/10.1007/978-3-030-35423-7_4

[43] 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.) EUROCRYPT 2016. vol. 9665, pp. 311–343. Springer (2016). https://doi.org/10.1007/978-3-662-49890-3_13

[44] Méaux, P., Park, J., Pereira, H.V.L.: Towards Practical Transciphering for FHE with Setup Independent of the Plaintext Space. Cryptology ePrint Archive, Paper 2023/1531 (2023), https://eprint.iacr.org/2023/1531, to appear in Communications in Cryptology

[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). https://doi.org/10.1145/2046660.2046682

[46] Saarinen, M.J.O.: Cryptographic Analysis of All 4 × 4-Bit S-Boxes. In: Miri, A., Vaudenay, S. (eds.) Selected Areas in Cryptography. pp. 118–133. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28496-0_7

[47] Yang, B.Y., Chen, J.M.: Theoretical Analysis of XL over Small Fields. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) Information Security and Privacy. pp. 277–288. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27800-9_24

[48] Yang, B.Y., Chen, O.C.H., Bernstein, D.J., Chen, J.M.: Analysis of QUAD. In: Biryukov, A. (ed.) FSE 2007. pp. 290–308. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74619-5_19

[49] Yang, Z., Xie, X., Shen, H., Chen, S., Zhou, J.: Tota: Fully homomorphic encryption with smaller parameters and stronger security. Cryptology ePrint Archive, Paper 2021/1347 (2021), https://eprint.iacr.org/2021/1347

[50] Zama: TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data (2022), https://github.com/zama-ai/tfhe-rs

[51] Zhang, W., Bao, Z., Rijmen, V., Liu, M.: A New Classification of 4-bit Optimal S-boxes and Its Application to PRESENT, RECTANGLE and SPONGENT. In: Leander, G. (ed.) FSE 2015. pp. 494–515. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48116-5_24

Supplementary Material

A Keyschedule Matrix of FRAST

The following is the invertible 64×64 matrix $\mathbf{M}$ over $\mathbb{Z}_{16}$ to generate the round keys for FRAST in hex. The j-th hex digit in the i-th row denotes $\mathbf{M}_{i,j}$.

```
[
    123e2d4f0a68befe1b7d2a8a858a011cc4f75074a6112874974ea0ed3f003e97,
    78c9a6ca9af337f6c16d9b8d9651790a672aff5a4ddf5ef53801e55d71111a3a,
    688b7691e958404437ef1190b84aae2a289ce0aeff6ddde8899e9db73770bfb0,
    b9dfd3ad6350b7b68b59249681939dea07d3e1897052989f651d8f902cbd2e6d,
    d88befa672f89f48ae8827afb8c57350d05d44f09f858ffb309c33e4fcbc2b7c,
    3390d1a59bbc7345f617dd5d1d242b7ca5d59fb237c28e91a4d1d9e03aa526ac,
    cf01a1143779cdca5a2d909144dad777a5f1bdf10d0b4355b46950aea6810c28,
    54db8d16b878595f09aaa5569293f1e16453923dd9d639df690d545cb34eb287,
    e33d69fe80adfd18b16f0885ce5a83e2cf08a35d785c22755a2aee03c5364d19,
    7733b2708ef22719a886ef7e83909fbff6ad122e08d876646f85ca5e737a9784,
    6b56dda75fa664d45b8eed98bfc10aa0a73c756b2826c88a5734beb6d458a635,
    7c764e4ee7ea41d2ab2e52e65dae1258633fab9f9824eeda7461bd1e5d9d5dfa,
    274929e465b09f096a890853505f4422e732d34cc6ad11df6b9a318834567981,
    43f45459c7eb531181f9069772106b4700a14264378d8a1188c519444dc412b4,
    56030bb64d3e51e69a52d0e6d835fb6fde71190fe0990c50506269b4fb53c50e,
    8ecece73caf1f42a311ed490388e62f5f51b878d1ad2c3cf49091b97e771428d,
    f798b7780d3528223e64577a4e93067bd70b94c563caa1b02843942c7eab2e0d,
    88208a47bd7348fd3711dc63fa4b51c3c78a124f147f13aecae38dcaec6434c6,
    4bd72876ca5df876a90965429c591904bcdc46b2472ca6a8a5a8a71611fc3a38,
    81145d688175a61195940fa86398e5b18f209595179454709aeade1e0206731a,
    5056684dc7d6598511e9a67f5f131f11e096754fc7fd992f0c9428121c2cab55,
    8045cd6a51f879e0ad1fac41eb4935f5393d64c4170b413de21a6862a162d5c8,
    81c639033f98b85f1b0da21c235971e1329c7bbca555d3d948fb8ec788f2ea92,
    d900297fad1be4ec5c1b941e486ad8d2378ab5150c31444665eda786407fafbe,
    80deccafb5db083163d050859df830017953c9e39bb3ac1f0af14d519538d16f,
    2ddbb2af637f17c67ff5ecee151edde07a3cccfee1ce610ef843d6f131502812,
    13bb3c0c389e5d1b64e51becd6b1a7ec80ba13eac4d90a81c87a316425249b7f,
    bd2de0f717cf1d713430da20fc4b425ec482a79ef9b360f9e55b71dbd44ad6f4,
    d16221d21e6889f3d4666f735aa2ca9bf02adee153cd79d64cd307015eab836a,
    31d26500d305b0ec48a681aba0de1a51b4433c2d50f19e74dff72d2e4147a655,
    64f97ff5ddf77ecc9af0b9a169158fba74928ad5ad8262dd300741d8ae686551,
    5e94a56cc8643dec58df4605bcbe2e15564bd2b59735efbab6ec7b0955e3c24a,
    6c5bcd35dd82eba1421f985324092acb190739956c86399f1bdcf1d060a8f583,
    32b43d7d456a82b38cb09b1af9b1a927934b8abb1120e48c1823bb3ce41bc3e6,
    67ef894f2d1c96525a8313fcae3ea323f7676a02d5c77276a62a9084a35171f8,
    6d91fb6ec05b7ebf023cc5f194c89252a1a7c57a23fed84801c9aa8730daf1d3,
    c22ea03a5e77a62e7a938d43b3aa3a2afb681fe7bbb585a3d0a13f4abfeb7235,
    8d1b6b34141a0681f984a3c1a531a3f89b90ca8f9500ee34c6ba15e3ae102a5a,
    68d0a550478363bc5f184798bd9eda8bead7fa2da1c8a05190619676138a9220,
    cc5d2c6e0fc017d51dd55f15487505e6c8e49c8d5254552bf99e9f027a157d0d,
    e46d369567e85730d69850b5a66381e8b351683333b7506ad2a7255b64d73d62,
    28c59e68559ee74a28b004315653ee7b60201141119be7f6c9c5db6a9426aa84,
    f8a619ee67f02c28240731427c7a73510de95c8d14bd535615124f7f2ea28536,
    08cc2655daa1166a3e1e27e4d32e1964f4668906105704f16674e05f454674ac,
    824b77fb8ebac0cfd6e501ebdd8399ffc6d3fdfaa252c546f07b2fcb3211c44b,
    338ee3c9b964f458368b12fb5dda0a21d71b82895f344c84d79a209d2a8a2e9e,
    11f895bcaa829a35766a9df798f76fdb75f52acf844b38e29843b3c4cba641e0,
    76d3864ea625fa5162e954ee2e21de7d2b31f5c2a848b055f9a8d2ce3c4d602a,
    8d2f7999216bb4ccb0ab4d2f3ea49216e3806f68911777716c5a3f57f39ad5ab,
    932be23c972a02d874c896c8bdd9f9e8f25c7597255c785febb36f39f21e6047,
    2af439cffb90a7e3cd449b98b4e41986c62feeeac893e948f6d39bccbaea882c,
    b2ce76fa91b65528128fc2f75cb210d627de1a60405a99fe68049adf26ce6290,
    90199dbacdbc4cd27d89078cb88488c953e8aafe40f9fdf145ac4126807ce2bb,
    1e9edf31bc7674add23dc865f2b7e459fce2855b725168b709246198b67c28bc,
    365c951b8f9157c59975fe4e2b2fe6d20559e8945cc951f2d66fa60189181705,
    6ae54a50a73127b6e384b823cd5f7274c499db4357aa984134fcb8d7a2263a1c,
    af24aab01b444a3a8d1959a200051f61b154411a9318d26f3c9fc9089dc5a15e,
    a7c8d71ff32d30c5da29787bc37f7faf3ec301db0ee66623d73a44cf0a5221fc,
    8db8ecc3b85043eb2c092b39393a501b286fe5133f2f13182b262c3fd5fb0ffe,
    824d35b5f8343f86fcb4c850113c7b1c57c500cbe3c7d3dd57c2fffeaa472d73,
    255bf17ef996b37e6a2031faeb3ebb5171db7b3a9edc299b1fc7b068acf41e67,
    123e3ef3ce478c12792754b8fc94ba37dc7331415b0e1ad8fef70b4002500580,
    fde6a170a8ba8349a6f366f26e5bdf1a4979c94ca8800c4875d3ef6e3314fac3,
    cecab9b85ef7b9fbe7792d72c43b144ccc1cabc24a3403c38e00d8763610b6bb
];
```

B Algebraic Representations of the **FRAST** Fixed S-box

Let $(x_0, \ldots, x_3) \in \mathbb{F}_2^4$ (resp. $(y_0, \ldots, y_3) \in \mathbb{F}_2^4$) be the input (resp. output) of S where x_0 (resp. y_0) is the LSB of the input (resp. output). The explicit representation of S over $\mathbb{F}_2$ is given as follows.

$$\begin{cases} y_0 = x_0x_1x_3 + x_0x_2x_3 + x_0 + x_1x_2 + x_1 + x_3 \\ y_1 = x_0x_1x_2 + x_0x_1x_3 + x_0x_1 + x_0 + x_1x_2x_3 + x_1x_2 + x_1x_3 + x_2x_3 + x_2 \\ y_2 = x_0x_1 + x_0x_2x_3 + x_0x_2 + x_0x_3 + x_1x_2x_3 + x_1x_2 + x_1x_3 + x_1 + x_2 + x_3 \\ y_3 = x_0x_1x_2 + x_0x_1 + x_0x_2 + x_1x_2 + x_2x_3 + x_3 \end{cases}$$

All the output bits are of degree 3 with respect to the input bits, and all the input bits work nonlinearly to the output as described in [46]. For the backward direction, we obtain the following equations which have the same properties with the forward ones.

$$\begin{cases} x_0 = y_0y_1y_3 + y_0y_2y_3 + y_0y_2 + y_1y_3 + y_1 + y_2 + y_3 \\ x_1 = y_0y_1y_3 + y_0y_1 + y_0y_3 + y_1y_2y_3 + y_1 + y_2y_3 + y_2 + y_3 \\ x_2 = y_0y_2y_3 + y_0y_2 + y_0 + y_1y_2y_3 + y_1y_3 + y_1 + y_2y_3 \\ x_3 = y_0y_1y_2 + y_0y_2y_3 + y_0y_3 + y_0 + y_1 + y_2y_3 + y_2 \end{cases}$$

The implicit representations over $\mathbb{F}_2$ are given as follows.

$$\begin{cases}
x_3x_2 + x_3y_3 + x_3y_1 + x_3y_0 = 0 \\
x_3x_2 + x_2x_1 + x_2x_0 + x_3y_2 + x_3y_0 + x_2 + x_1 + y_3 + y_1 + y_0 = 0 \\
x_3x_2 + x_2y_3 + x_3y_2 + x_3y_0 + x_2 + x_1 + y_3 + y_1 + y_0 = 0 \\
x_2x_1 + x_3x_0 + x_2y_2 + x_2 + y_2 + y_1 + y_0 = 0 \\
x_3x_2 + x_3x_1 + x_3x_0 + x_3y_3 + x_3y_2 + x_2y_1 + x_1 + y_3 + y_1 + y_0 = 0 \\
x_3x_1 + x_3y_3 + x_2y_0 + x_2 + x_0 + y_3 + y_1 = 0 \\
x_3x_0 + x_1x_0 + x_3 + y_3 + y_2 + y_1 + y_0 = 0 \\
x_1y_3 + x_3y_0 + x_3 + x_1 + x_0 + y_3 + y_2 + y_1 = 0 \\
x_2x_1 + x_3x_0 + x_3y_3 + x_1y_2 + x_3y_0 + x_2 + x_1 + x_0 + y_1 = 0 \\
x_3x_2 + x_3x_1 + x_1y_1 + x_3y_0 + x_3 + x_2 + x_1 + y_3 + y_2 = 0 \\
x_2x_1 + x_3x_0 + x_3y_3 + x_3y_2 + x_3y_0 + x_1y_0 + x_1 + y_3 + y_2 + y_1 + y_0 = 0 \\
x_3x_0 + x_0y_3 + x_3y_0 + x_3 + x_2 + x_0 + y_2 + y_0 = 0 \\
x_3x_2 + x_3x_1 + x_2x_1 + x_3y_3 + x_3y_2 + x_0y_2 + x_3 + x_2 + x_0 + y_3 + y_2 + y_0 = 0 \\
x_3x_2 + x_3x_1 + x_3x_0 + x_3y_3 + x_0y_1 + x_3 + x_2 + y_2 + y_0 = 0 \\
x_2x_1 + x_3x_0 + x_3y_2 + x_3y_0 + x_0y_0 + x_2 + y_1 = 0 \\
x_3x_2 + x_3x_1 + x_3x_0 + x_3y_2 + y_3y_2 + x_3 + x_2 + x_0 + y_3 + y_2 + y_0 = 0 \\
x_3x_1 + x_3x_0 + y_3y_1 = 0 \\
x_3x_1 + y_3y_0 + x_1 + x_0 + y_2 + y_1 = 0 \\
x_3x_2 + x_3x_1 + x_3x_0 + y_2y_1 + x_3y_0 + x_3 + x_1 + y_3 + y_1 + y_0 = 0 \\
x_3x_2 + x_2x_1 + y_2y_0 + x_1 + y_3 + y_2 + y_1 + y_0 = 0 \\
x_3x_1 + y_1y_0 + x_2 + y_2 + y_0 = 0
\end{cases}$$

There are 21 linearly independent quadratic equations for S, which is the minimum for 4-bit S-boxes.

C The Number of Monomials on Toy Parameters

In this section, we show the experimental result of the number of monomials appearing in the system of equations for the XOR-variant of FRAST in the key variables. We used toy parameters of $\ell = 4$ and the key size of 4ℓ bits where ℓ is the number of branches.

We chose 100 random inputs, represented the corresponding outputs for a single and two fixed rounds of the XOR-variant of FRAST as polynomials in the key variables, and counted the number of monomials according to their degrees. Table 4 summarize the results. One can see that more than 90% of monomials of degree d appear in the system where d is the lower bound proposed in Section 5.1.1.

We also experimented in the backward direction considering the meet-in-the-middle attack, and summarized the results in Table 5. One can see that more than 90% of monomials of degree d appear after two rounds in the backward direction where d is the lower bound proposed in Section 5.1.1.

Degree	Total	Single Round				Two Rounds			
		First	Ratio	Others	Ratio	First	Ratio	Others	Ratio
1	16	15.24	0.953	16.00	1.000	14.91	0.932	16.00	1.000
2	120	112.74	0.940	119.80	0.998	112.52	0.938	119.93	0.999
3	560	524.99	0.937	**542.68**	**0.969**	525.45	0.938	559.78	0.999
4	1820	1707.71	0.938	-	-	1705.61	0.937	1819.62	0.999
5	4368	**4091.44**	**0.937**	-	-	4093.88	0.937	4367.08	0.999
6	8008	7485.90	0.935	-	-	7506.56	0.937	8006.04	0.999
7	11440	9989.81	0.873	-	-	10725.53	0.938	**11437.21**	**0.999**
8	12870	10852.44	0.843	-	-	12068.20	0.938	12866.73	0.999
9	11440	5254.97	0.459	-	-	**10725.86**	**0.938**	11428.58	0.999
10	8008	-	-	-	-	7505.68	0.937	7863.41	0.982
11	4368	-	-	-	-	4095.01	0.938	3930.39	0.900
12	1820	-	-	-	-	1705.20	0.937	-	-
13	560	-	-	-	-	524.46	0.937	-	-
14	120	-	-	-	-	112.95	0.941	-	-
15	16	-	-	-	-	14.17	0.886	-	-
16	1	-	-	-	-	0.89	0.890	-	-

Table 4: The (average) number of monomials appearing in the fixed round functions of FRAST in the forward direction according to degrees. 'Total' denotes the total number of monomials, 'First' (resp. 'Others') denotes the number of monomials appearing in the first branch (resp. the other branches), and 'Ratio' denotes its ratio to the total number of monomials. The bold fonts denote the lower bound of the degree proposed in Section 5.1.1.

Degree	Total	Single Round				Two Rounds			
		First	Ratio	Others	Ratio	First	Ratio	Others	Ratio
1	16	16.00	1.000	16.00	1.000	14.97	0.936	16.00	1.000
2	120	108.07	0.901	119.92	0.999	112.53	0.938	119.96	0.999
3	560	**409.45**	**0.731**	558.90	0.998	524.48	0.937	559.88	0.999
4	1820	-	-	1815.19	0.997	1706.24	0.937	1819.42	0.999
5	4368	-	-	**4213.67**	**0.965**	4096.11	0.938	4366.93	0.999
6	8008	-	-	-	-	7504.14	0.937	8005.58	0.999
7	11440	-	-	-	-	**10718.37**	**0.937**	11437.00	0.999
8	12870	-	-	-	-	12049.91	0.936	12866.47	0.999
9	11440	-	-	-	-	9945.95	0.869	**11436.99**	**0.999**
10	8008	-	-	-	-	6671.91	0.833	8006.18	0.999
11	4368	-	-	-	-	1987.14	0.455	4366.63	0.999
12	1820	-	-	-	-	-	-	1810.18	0.995
13	560	-	-	-	-	-	-	529.75	0.946

Table 5: The (average) number of monomials appearing in the fixed round functions of FRAST in the backward direction according to degrees. 'Total' denotes the total number of monomials, 'First' (resp. 'Others') denotes the number of monomials appearing in the first branch (resp. the other branches), and 'Ratio' denotes its ratio to the total number of monomials. The bold fonts denote the lower bound of the degree proposed in Section 5.1.1.

D Gröbner Basis Computation on Toy Parameters

In this section, we summarize the experimental result of the Gröbner basis computation time on toy parameters. The source codes of the experiment are developed in MAGMA [13], and are executed in AMD Ryzen 7 2700X @ 3.70 GHz with 128 GB memory. Let r, ℓ, and m denote the number of rounds, branches, and input/output pairs used to build a system, respectively.

Figure 5 shows the Gröbner basis computation time according to r when $\ell = 3$ for the system introducing new variables for all the intermediate states and the system introducing new variables for the states of the first branch. The key size is set to 4ℓ, which is half of the actual key size, to run the experiment on various parameters. One can see that the computation time grows exponentially according to r, and using larger m increases computation time.

Figure 6 shows the Gröbner basis computation time for the system by guessing the values of the first branch according to ℓ when $r = 8$. The key size is set to 8ℓ, which is the actual key size[21]. One can see that the computation time grows exponentially according to ℓ. The peak at $(\ell, m) = (3, 3)$ is caused by some outliers in the data.

Conversely to the previous systems, using larger m tends to decrease the Gröbner basis computation time. The reason is that the number of variables

[21] The graph in Figure 4 uses the key size of 4ℓ.

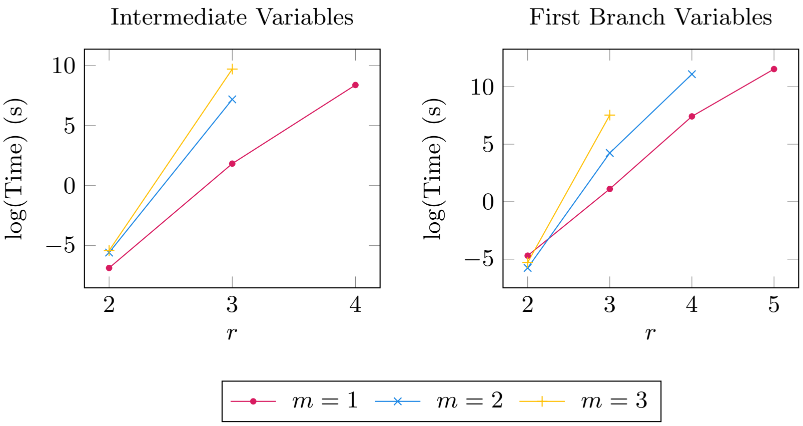


Fig. 5: Gröbner basis computation time of the systems introducing new variables according to the number of rounds r. The number of branches ℓ is 3 and the key size is 4ℓ bits.

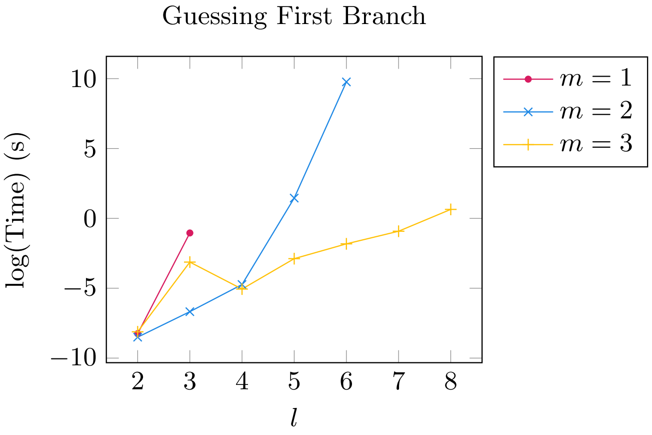


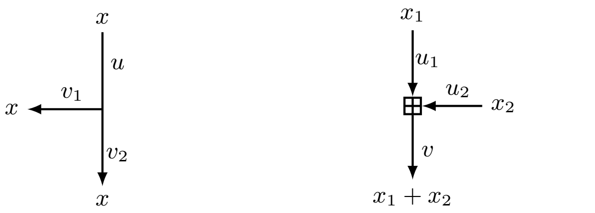
Fig. 6: Gröbner basis computation time of the systems with guessing the values of the first branches. The number of round r is 8 and the key size is 8ℓ bits.

does not change while the number of equations increases according to m. That said, this does not imply using larger m is advantageous for the attack since the number of guessing 2^{4rm} increases much faster.

Remark 3. The whole running time of the Gröbner basis computing program is much longer than the Gröbner basis computing time denoted in the graph, so we could not run the experiment on larger parameters.

E Linear Cryptanalysis on FRAST

In this section, we describe the linear cryptanalysis on FRAST in detail. Before going into the details, we recap the condition on input/output linear masks for some linear operations. For a branching operation $x \mapsto (x, x)$ with input mask u and output mask (v_1, v_2), $u = v_1 + v_2$ should be satisfied. For an addition operation $(x_1, x_2) \mapsto x_1 + x_2$ with input mask (u_1, u_2) and output mask v, $u_1 = u_2 = v$ should be satisfied. See Figure 7.



(a) Branching: $u = v_1 + v_2$ (b) Addition: $u_1 = u_2 = v$

Fig. 7: Linear masks for branching and addition

E.1 Analysis with Linear Masks on XOF Outputs

The first approach is to apply nonzero linear masks on the XOF outputs that determine random S-boxes of FRAST. Although the XOF outputs themselves are not controllable by an attacker, they can be considered as additional inputs in the KPA model since they are publicly open. Using this approach, one can apply the linear cryptanalysis to FRAST by considering it as a fixed function whose input size is larger than its output size. A negacyclic random S-box S itself can be described by its function values $(S(0), \ldots, S(7))$, and the function $\mathsf{LookUp}_k(x, S) = S(x + k)$ can be described as a function from $\mathbb{Z}_{16} \times \mathbb{Z}_{16}^8$ to $\mathbb{Z}_{16}$ defined by

$$\mathsf{LookUp}_k(x, S) = \sum_{i=0}^{7} \left(\mathbf{1}\left\{x + k = i\right\} - \mathbf{1}\left\{x + k = i + 8\right\} \right) S(i)$$

where $\mathbf{1}\{x+k=i\}=1$ if $x+k=i$ and 0 otherwise. From this point of view, the linear probability of FRAST with additional linear masks on the XOF outputs is well-defined.

For $(\mathbf{x}, S_{\mathsf{erf}}, S_{\mathsf{crf}}) \in \mathbb{Z}_{16}^{\ell} \times \mathbb{Z}_{16}^{8} \times \mathbb{Z}_{16}^{8}$, define $\mathsf{RF}[\mathbf{k}](\mathbf{x}, S_{\mathsf{erf}}, S_{\mathsf{crf}}) = \mathbf{y} \in \mathbb{Z}_{16}^{\ell}$ as follows.

$$\begin{aligned} y_j &= x_j + \mathsf{LookUp}_{rk_j}(x_1, S_{\mathsf{erf}}) \text{ for } j = 2, \dots, \ell, \\ y_1 &= x_1 + \mathsf{LookUp}_{rk_1}(y_2 + \dots + y_\ell, S_{\mathsf{crf}}), \end{aligned}$$

where $\mathbf{rk} = (rk_1, \dots, rk_\ell) \in \mathbb{Z}_{16}^{\ell}$ is a round key derived from the master key $\mathbf{k}$, $\mathbf{x} = (x_1, \dots, x_\ell) \in \mathbb{Z}_{16}^{\ell}$, $\mathbf{y} = (y_1, \dots, y_\ell) \in \mathbb{Z}_{16}^{\ell}$, and S_{erf} and S_{crf} are the negacyclic S-boxes derived from the XOF. As S_{erf} and S_{crf} are independently sampled, we separate the round function RF into two parts for simplicity: $\mathsf{RF}_{\mathsf{erf}}$ and $\mathsf{RF}_{\mathsf{crf}}$.

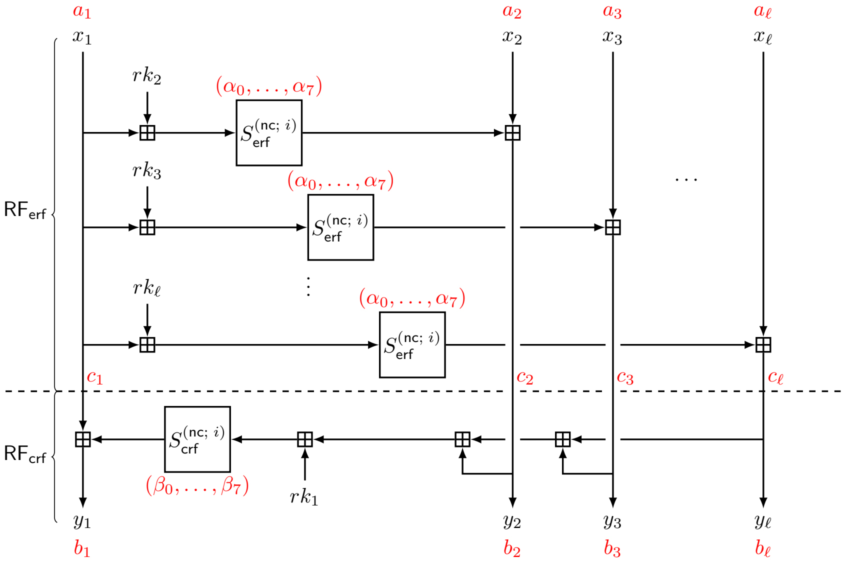

a_1
x_1
a_2
x_2
a_3
x_3
a_ℓ
x_ℓ
rk_2
$(\alpha_0, \dots, \alpha_7)$
$S_{\mathsf{erf}}^{(\mathsf{nc};\, i)}$
rk_3
$(\alpha_0, \dots, \alpha_7)$
$S_{\mathsf{erf}}^{(\mathsf{nc};\, i)}$
$\cdots$
$\mathsf{RF}_{\mathsf{erf}}$
$\vdots$
rk_ℓ
$(\alpha_0, \dots, \alpha_7)$
$S_{\mathsf{erf}}^{(\mathsf{nc};\, i)}$
c_1
c_2
c_3
c_ℓ
$\mathsf{RF}_{\mathsf{crf}}$
$S_{\mathsf{crf}}^{(\mathsf{nc};\, i)}$
$(\beta_0, \dots, \beta_7)$
rk_1
y_1
b_1
y_2
b_2
y_3
b_3
y_ℓ
b_ℓ

Fig. 8: Linear masks in a single round of the FRAST considering XOF.

$\mathsf{RF}_{\mathsf{erf}}$ is the expanding part of the round function. For $(\mathbf{x}, S_{\mathsf{erf}}) \in \mathbb{Z}_{16}^{\ell} \times \mathbb{Z}_{16}^{8}$, $\mathsf{RF}_{\mathsf{erf}}[\mathbf{k}](\mathbf{x}, S_{\mathsf{erf}}) = \mathbf{y} \in \mathbb{Z}_{16}^{\ell}$ is defined as follows.

$$\begin{aligned} y_j &= x_j + \mathsf{LookUp}_{rk_j}(x_1, S_{\mathsf{erf}}) \text{ for } j = 2, \dots, \ell, \\ y_1 &= x_1, \end{aligned}$$

where $(rk_2, \dots, rk_\ell) \in \mathbb{Z}_{16}^{\ell-1}$ is a round key derived from the master key $\mathbf{k}$, $\mathbf{x} = (x_1, \dots, x_\ell) \in \mathbb{Z}_{16}^{\ell}$, $\mathbf{y} = (y_1, \dots, y_\ell) \in \mathbb{Z}_{16}^{\ell}$, and S_{erf} is the negacyclic S-box derived from the XOF.

$\mathsf{RF}_{\mathsf{crf}}$ is the contracting part of the round function. For $(\mathbf{x}, S_{\mathsf{crf}}) \in \mathbb{Z}_{16}^{\ell} \times \mathbb{Z}_{16}^{8}$, $\mathsf{RF}_{\mathsf{crf}}[\mathbf{k}](\mathbf{x}, S_{\mathsf{crf}}) = \mathbf{y} \in \mathbb{Z}_{16}^{\ell}$ is defined as follows.

$$\begin{aligned} &y_j = x_j \text{ for } j = 2, \ldots, \ell, \\ &y_1 = x_1 + \mathsf{LookUp}_{rk_1}(x_2 + \cdots + x_\ell, S_{\mathsf{crf}}), \end{aligned}$$

where $rk_1 \in \mathbb{Z}_{16}$ is a round key derived from the master key $\mathbf{k}$, $\mathbf{x} = (x_1, \ldots, x_\ell) \in \mathbb{Z}_{16}^{\ell}$, $\mathbf{y} = (y_1, \ldots, y_\ell) \in \mathbb{Z}_{16}^{\ell}$, and S_{crf} is the negacyclic S-box derived from the XOF.

Then, the round function RF can be described as a composition of $\mathsf{RF}_{\mathsf{erf}}$ and $\mathsf{RF}_{\mathsf{crf}}$ as follows.

$$\mathsf{RF}[\mathbf{k}](\mathbf{x}, S_{\mathsf{erf}}, S_{\mathsf{crf}}) = \mathsf{RF}_{\mathsf{crf}}[\mathbf{k}]\left(\mathsf{RF}_{\mathsf{erf}}[\mathbf{k}](\mathbf{x}, S_{\mathsf{erf}}), S_{\mathsf{crf}}\right).$$

We depict the relation between RF, $\mathsf{RF}_{\mathsf{erf}}$ and $\mathsf{RF}_{\mathsf{crf}}$ in Figure 8. By separating RF into $\mathsf{RF}_{\mathsf{erf}}$ and $\mathsf{RF}_{\mathsf{crf}}$, we can compute the linear probability of RF from that of $\mathsf{RF}_{\mathsf{erf}}$ and $\mathsf{RF}_{\mathsf{crf}}$.

Let $\mathbf{a} = (a_1, \ldots, a_\ell)$ be an input mask to $\mathbf{x}$, $\boldsymbol{\alpha} = (\alpha_0, \ldots, \alpha_7)$ (resp. $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_7)$) be an input mask to S_{erf} (resp. S_{crf}), and $\mathbf{b} = (b_1, \ldots, b_\ell)$ be an output mask to $\mathbf{y}$. To represent the linear probability of RF with respect to those of $\mathsf{RF}_{\mathsf{erf}}$ and $\mathsf{RF}_{\mathsf{crf}}$, let $\mathbf{c} = (c_1, \ldots, c_\ell)$ be an output (resp. input) mask of $\mathsf{RF}_{\mathsf{erf}}$ (resp. $\mathsf{RF}_{\mathsf{crf}}$) (see Figure 8). Then, the linear relation on RF forces $c_j = a_j$ for all $j = 2, \ldots, \ell$, $c_1 = b_1$, and $c_2 - b_2 = \cdots = c_\ell - b_\ell$. For such $\mathbf{c}$, we obtain the following.

$$\mathsf{LP}^{\mathsf{RF}[\mathbf{k}]}((\mathbf{a}, \boldsymbol{\alpha}, \boldsymbol{\beta}), \mathbf{c}) = \mathsf{LP}^{\mathsf{RF}_{\mathsf{erf}}[\mathbf{k}]}((\mathbf{a}, \boldsymbol{\alpha}), \mathbf{c}) \cdot \mathsf{LP}^{\mathsf{RF}_{\mathsf{crf}}[\mathbf{k}]}((\mathbf{c}, \boldsymbol{\beta}), \mathbf{b}).$$

Linear Probability of $\mathsf{RF}_{\mathsf{erf}}$. Let $(\mathbf{a}, \boldsymbol{\alpha}) \in \mathbb{Z}_{16}^{\ell} \times \mathbb{Z}_{16}^{8}$ be the input mask and $\mathbf{c} \in \mathbb{Z}_{16}^{\ell}$ be the output mask to $\mathsf{RF}_{\mathsf{erf}}[\mathbf{k}]$ where $c_j = a_j$ for $j = 2, \ldots, \ell$. Then, the linear probability of $\mathsf{RF}_{\mathsf{erf}}[\mathbf{k}]$ is given as follows.

$$\begin{aligned} \mathsf{LP}^{\mathsf{RF}_{\mathsf{erf}}[\mathbf{k}]}((\mathbf{a}, \boldsymbol{\alpha}), \mathbf{c}) = \frac{1}{16^2} \Bigg| &\sum_{x_1=0}^{15} \exp\left(\frac{2\pi i}{16}(c_1 - a_1)x_1\right) \\ &\times \mathbf{1}\left\{ {}^{\forall j \in \{0,\ldots,7\}}_{\sum_{i=2}^{\ell} c_i (\mathbf{1}\{x_1 + rk_i = j\} - \mathbf{1}\{x_1 + rk_i = j+8\}) = \alpha_j} \right\} \Bigg|^2 \end{aligned}$$

where $(rk_2, \ldots, rk_\ell)$ is a part of the round key derived from $\mathbf{k}$. One can see that the above linear probability depends on the relation between the masks and the keys, which is not the case in traditional linear cryptanalysis. Hence, the masks should be chosen carefully to satisfy the following relation.

$$\exists x_1 \in \mathbb{Z}_{16};\ \sum_{i=2}^{\ell} c_i \left(\mathbf{1}\left\{x_1 + rk_i = j\right\} - \mathbf{1}\left\{x_1 + rk_i = j + 8\right\}\right) = \alpha_j \ \forall j = 0, \ldots, 7. \tag{10}$$

Otherwise, the linear probability would be zero.

For an attacker who does not know the round key, there are two possible ways to build a trail of nonzero linear probability: a trivial linear trail such that $\boldsymbol{\alpha} = \mathbf{0}$, and a linear trail such that only one component of $\boldsymbol{\alpha}$ is nonzero.

The first approach is to build a linear trail that does not activate the LookUp function by setting $\boldsymbol{\alpha} = \mathbf{0}$ and $c_2 = \cdots = c_\ell = 0$, which also implies that $a_2 = \cdots = a_\ell = 0$. Then, by setting $c_1 = a_1 \neq 0$, one obtain $\mathsf{LP}^{\mathsf{RF}_{\mathsf{erf}}[\mathbf{k}]}((\mathbf{a}, \boldsymbol{\alpha}), \mathbf{c}) = 1$ for nonzero input/output masks. This is the trivial linear trail of linear probability 1 on $\mathsf{RF}_{\mathsf{erf}}$.

The other approach is to set only one component of $(c_2, \ldots, c_\ell)$ and $\boldsymbol{\alpha}$ to 8 and the others to 0. Then, regardless of the round key, there exists a unique $z \in \{0, \ldots, 7\}$ such that (10) holds for $x_1 = z$ and $x_1 = z + 8$. By setting $2 \mid (c_1 - a_1)$, one obtain $\mathsf{LP}^{\mathsf{RF}_{\mathsf{erf}}[\mathbf{k}]}((\mathbf{a}, \boldsymbol{\alpha}), \mathbf{c}) = 2^{-6}$.

Linear Probability of $\mathsf{RF}_{\mathsf{crf}}$. Let $(\mathbf{c}, \boldsymbol{\beta}) \in \mathbb{Z}_{16}^{\ell} \times \mathbb{Z}_{16}^{8}$ be the input mask and $\mathbf{b} \in \mathbb{Z}_{16}^{\ell}$ be the output mask to $\mathsf{RF}_{\mathsf{crf}}[\mathbf{k}]$ where $c_1 = b_1$ and $c_2 - b_2 = \cdots = c_\ell - b_\ell$. Then the linear probability of $\mathsf{RF}_{\mathsf{crf}}[\mathbf{k}]$ is given as follows.

$$\begin{aligned}\mathsf{LP}^{\mathsf{RF}_{\mathsf{crf}}[\mathbf{k}]}((\mathbf{c}, \boldsymbol{\beta}), \mathbf{b}) = \frac{1}{16^2} \Bigg| \sum_{x=0}^{15} & \exp\left(\frac{2\pi i}{16}(b_2 - c_2)x\right) \\ & \times \mathbf{1}\left\{ \substack{\forall j \in \{0,\ldots,7\} \\ b_1(\mathbf{1}\{x + rk_1 = j\} - \mathbf{1}\{x + rk_1 = j+8\}) = \beta_j} \right\} \Bigg|^2\end{aligned}$$

where rk_1 is the first component of the round key derived from $\mathbf{k}$. The masks should be chosen carefully to satisfy the following to build a linear trail of nonzero linear probability.

$$\exists x \in \mathbb{Z}_{16};\ b_1 \left(\mathbf{1}\left\{x + rk_1 = j\right\} - \mathbf{1}\left\{x + rk_1 = j + 8\right\}\right) = \beta_j \ \forall j = 0, \ldots, 7. \quad (11)$$

Similar to the case of $\mathsf{RF}_{\mathsf{erf}}$, an attacker can build two kinds of linear trails of nonzero linear probability without knowing the round key. One is the trivial linear trail to set $\boldsymbol{\beta} = \mathbf{0}$ and $b_1 = c_1 = 0$, obtaining $\mathsf{LP}^{\mathbf{rk}_{\mathsf{crf}}[\mathbf{k}]}((\mathbf{b}, \boldsymbol{\beta}), \mathbf{b}) = 1$. The other nontrivial trail is to set $b_1 = 8$, only one component of $\boldsymbol{\beta}$ by 8 and the others by 0, obtaining the linear probability of 2^{-6} provided that $2 \mid (c_2 - b_2)$.

Combining two results. One can build a linear trail on RF by combining those on $\mathsf{RF}_{\mathsf{erf}}$ and $\mathsf{RF}_{\mathsf{crf}}$. However, combining two trivial trails on $\mathsf{RF}_{\mathsf{erf}}$ and $\mathsf{RF}_{\mathsf{crf}}$ is impossible since it implies that all the input/output masks are zero. Instead, it is possible to combine one of the trivial trails and the other nontrivial trail, resulting in the linear trail of linear probability 2^{-6}. Such trail activates only one LookUp function.

If more than two LookUp functions are activated, then the attacker should know the difference between the round keys used in the activated LookUp functions. The attacker might try to guess them, but it is infeasible since more than 128 bits need to be guessed for 32 random rounds of FRAST.

E.2 Analysis on Compatible Data

The linear attack is a kind of statistical attack that requires many input-output pairs of a fixed function. In FRAST, negacyclic S-boxes are independent randomly selected. Hence, we need to compute the probability that two input-output pairs of FRAST from independent random round functions can be used together to measure the linear bias for a given linear approximation.

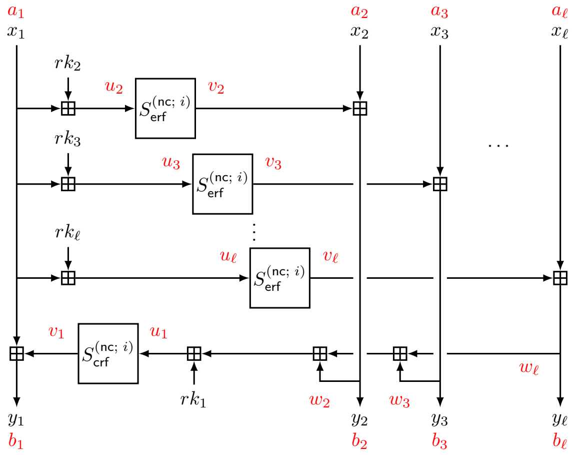


Fig. 9: Linear masks in a single round of the FRAST without considering XOF.

Linear Masks for the FRAST Round Function. Suppose that an input linear mask $\mathbf{a} = (a_1, a_2, \ldots, a_\ell)$ and an output linear mask $\mathbf{b} = (b_1, b_2, \ldots, b_\ell)$ are used for a single round of FRAST. Let u_i denote an input mask and let v_i denote an output mask of an S-box whose output is added to x_i (see Figure 9). From the properties of the branching and addition operations, the following conditions must hold for the input and output masks to have a nonzero linear probability.

$$v_1 = b_1 \text{ and } v_i = a_i \text{ for } i = 2, 3, \ldots, \ell, \tag{12}$$

$$u_1 = w_j = a_j - b_j \text{ for all } j = 2, 3, \ldots, \ell. \tag{13}$$

Suppose that there are nonzero input and output masks $\mathbf{a} = (a_1, a_2, \ldots, a_\ell)$ and $\mathbf{b} = (b_1, b_2, \ldots, b_\ell)$ activating no S-box. Since all the S-boxes are not activated, we have $b_1 = 0$ and $a_2 = a_3 = \cdots = a_\ell = 0$ by (12). We also have $b_2 = b_3 = \cdots = b_\ell$ by (13). Then the linear probability $\mathsf{LP}^{\mathsf{RF}}(\mathbf{a}, \mathbf{b})$ of the round function for the masks $\mathbf{a}$ and $\mathbf{b}$ is given as follows.

$$\begin{aligned}
\mathsf{LP}^{\mathsf{RF}}(\mathbf{a},\mathbf{b}) = \frac{1}{16^{2\ell}} &\left| \sum_{x_2,\cdots,x_\ell \in \mathbb{Z}_{16}} \exp\left\{ \frac{2\pi i}{16} \left(b_2\left(x_2+\cdots+x_\ell\right)\right) \right\} \right|^2 \\
&\times \left| \sum_{x_1 \in \mathbb{Z}_{16}} \exp\left\{ \frac{2\pi i}{16} \left(b_2 \sum_{j=2}^{\ell} S_{\mathsf{erf}}^{(\mathsf{nc};\, i)}(x_1 + rk_j) - a_1 x_1 \right) \right\} \right|^2 \\
&= \mathbf{1}\left\{a_1 = b_2 = 0\right\}
\end{aligned}$$

where $\mathbf{1}\{a_1 = b_2 = 0\}$ is 1 if $a_1 = b_2 = 0$ and 0 otherwise, $S_{\mathsf{erf}}^{(\mathsf{nc};\, i)}$ is the randomly generated S-box used in the round function and rk_j is the round key added to the input of the S-box whose output is added to the j-th branch. So we conclude that the input and output masks activating no S-box cannot be both nonzero and trivial.

Compatibility of a Linear Trail. Even for a round function using different negacyclic S-boxes, we can estimate its linear bias using the following condition: given two input-output pairs, suppose that their i-th round functions use S-boxes S_1 and S_2, respectively. Since S_1 and S_2 work identically with respect to the output masks, one can use both pairs together to see if $b(S_1(x) - S_2(x))$ is constant for all $x \in \{0, 1, \ldots, 7\} \subset \mathbb{Z}_{16}$ and for every nonzero output mask b for these negacyclic S-boxes. When S_1 and S_2 satisfy this condition, we say that S_1 and S_2 are *compatible* with respect to the output mask $b \neq 0$. If all the active S-boxes in the round functions are compatible with respect to their output masks, then we say that the round functions are compatible.

For independently sampled S_1 and S_2, we can compute the probability that these two S-boxes are compatible with respect to b. The probability is maximized when $b = 8$, which is 2^{-7}. The compatibility with respect to the output mask of 8 means that the LSBs of the outputs of the two S-boxes are either identical or all different. Since every pair of nontrivial input and output masks for the FRAST round function has at least one active S-box, the probability that two round functions are compatible is also upper bounded by 2^{-7}.

Now, consider a linear trail $T = (\mathbf{a}^{(i)})_{i=0}^{r}$ for r random rounds of FRAST, where $\mathbf{a}^{(i)} = (a_1^{(i)}, a_2^{(i)}, \ldots, a_\ell^{(i)}) \in \mathbb{Z}_{16}^{\ell}$ for $i = 0, 1, \ldots, r$. So $\mathbf{a}^{(i-1)}$ and $\mathbf{a}^{(i)}$ become an input and an output mask for the i-th round function, respectively, and the linear probability of the trail is given by the product of $\mathsf{LP}(\mathbf{a}^{(i-1)}, \mathbf{a}^{(i)})$ for all $i = 1, 2, \ldots, r$. To apply the trail on two input-output pairs, every round function for the two pairs should be compatible with respect to the output masks obtained by the trail. In this case, we say the two pairs are compatible with respect to the trail. The probability that the two pairs are compatible is upper bounded by 2^{-7r}, hence FRAST achieves 128-bit security against the linear attack if it has at least 19 random rounds.

E.3 Analysis with Random S-boxes Having Linear Relations

Suppose a negacyclic S-box S on $\mathbb{Z}_{16}$ has a linear relation $ax + bS(x) = c$ for all $x = 0, 1, \ldots, 15$. Then, the following holds from the negacyclic property of S.

$$a(x+8) + bS(x+8) = ax + 8a - bS(x) = c$$

over $\mathbb{Z}_{16}$ for all $x = 0, 1, \ldots, 7$. Combined with the original linear relation, one obtain

$$(2x+8)a = 2c$$

over $\mathbb{Z}_{16}$ for all $x = 0, 1, \ldots, 7$. It can holds only if $a, c \in \{0, 8\}$. In this case, the function $bS(x)$ should be identical to one of 0, 8, $8x$, and $8x + 8$, which implies that $bS(0)$ and $bS(1)$ determines $bS(i)$ for $i = 2, 3, \ldots, 7$. Hence, a negacyclic S-box over $\mathbb{Z}_{16}$ has a linear relation with a probability at most 2^{-6}.

F Function Decomposition of New WoP-PBS

For a t-bit integer $m \in [\![0, 2^t[\![$, let $\mathsf{FlipMSB}_t(m) = (m + 2^{t-1}) \bmod 2^t \in [\![0, 2^t[\![$. Given an arbitrary function f on $[\![0, p[\![$ for a power-of-two p, f can be decomposed into $\log p$ functions $f_0, f_1, \ldots, f_{\log p - 1}$ and one constant $f_{\log p}$ such that

$$f(x) = \sum_{j=0}^{\log p - 1} f_j(x \bmod p/2^j) + f_{\log p}$$

where f_j is a negacyclic function on $[\![0, p/2^j[\![$ defined as

$$f_j(x) = \frac{1}{2^{j+1}} \sum_{a=0}^{2^j - 1} \left(f(a \cdot 2^j + x) - f(a \cdot 2^j + \mathsf{FlipMSB}_{\log p - j}(x)) \right)$$

for $j = 0, \ldots, \log p - 1$, and $f_{\log p}$ is the constant given by

$$f_{\log p} = \frac{1}{p} \sum_{a=0}^{p-1} f(a).$$

This decomposition can be obtained by applying the decomposition of f into $\frac{1}{2}(f_{\textsf{msb-odd}} + f_{\textsf{msb-even}})$ described in Section 4.1 recursively on $f_{\textsf{msb-even}}$.

For example, let f be a function defined on $[\![0, 4[\![$. Then f is decomposed into two functions f_0, f_1 and one constant f_2. The function f_0 on $[\![0, 4[\![$ is given by

$$\begin{aligned} f_0(0) &= \frac{1}{2}(f(0) - f(2)), \\ f_0(1) &= \frac{1}{2}(f(1) - f(3)), \\ f_0(2) &= \frac{1}{2}(f(2) - f(0)), \\ f_0(3) &= \frac{1}{2}(f(3) - f(1)), \end{aligned}$$

the function f_1 on $\llbracket 0, 2 \llbracket$ is given by

$$f_1(0) = \frac{1}{4}(f(0) + f(2) - f(1) - f(3)),$$
$$f_1(1) = \frac{1}{4}(f(1) + f(3) - f(0) - f(2)),$$

and the constant f_2 is given by

$$f_2 = \frac{1}{4}(f(0) + f(1) + f(2) + f(3)).$$

G Communication Overload

In this section, we describes the communication overload of FRAST in the transciphering framework. The communication overload of the transciphering with TFHE consists of two parts: one for the homomorphic ciphertexts of the secret key, and the other for the TFHE evaluation keys only used for the transciphering. Since FRAST uses GLWEtoGGSW conversion in the setup phase, the evaluation keys for the conversion become additional communication overload.

Ciphertext Size. An LWE ciphertext $(a_1, \ldots, a_n, b) \in \mathbb{Z}_q^{n+1}$ consists of $n+1$ elements in $\mathbb{Z}_q$, so its size is given by $(n+1)\log q$ bits. If the LWE ciphertext is a fresh ciphertext such that no homomorphic operation is performed on it yet, the one can compress the random mask a into a seed for generating it. Such LWE ciphertexts are called seeded LWE ciphertexts. Ignoring the seed size by assuming that one seed generates all the random masks for multiple seeded ciphertexts, the size of the seeded LWE ciphertext is only $\log q$.[22] In case of a GLWE ciphertext $(A_1, \ldots, A_k, B) \in \mathcal{R}_{q,N}^{k+1}$, it is size of $(k+1)N\log q$ bits. When it compressed similarly, the seeded GLWE ciphertext is of size $N \log q$ bits. For a GGSW ciphertext $\mathbf{C} \in \mathcal{R}_{q,N}^{\ell(k+1)\times(k+1)}$, it can be considered as a vector of $\ell(k+1)$ GLWE ciphertexts. The size of a GGSW ciphertext is $\ell(k+1)^2 N \log q$ bits, and that of a compressed GGSW ciphertext is $\ell(k+1)N\log q$ bits. Table 6 summarizes the size of each type of TFHE ciphertexts.

	LWE	GLWE	GGSW
Normal	$(n+1)\log q$	$(k+1)N\log q$	$\ell(k+1)^2 N\log q$
Seeded	$\log q$	$N\log q$	$\ell(k+1)N\log q$

Table 6: Size of TFHE ciphertexts in bits. The size of seeds or auxiliary information is ignored.

[22] In the `tfhe-rs` library, auxiliary information such as the LWE dimension or ciphertext modulus type is saved together. We ignore such additional data size assuming that it is fixed in the transciphering framework.

GenPBS Keysize. The evaluation keys for the GenPBS operation consist of the bootstrapping key and the keyswitching key. Given an LWE secret key $\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{B}^n$ and a GLWE secret key $\mathbf{S}' = (S'_1, \ldots, S'_k) \in \mathbb{B}_N[X]^k$, the bootstrapping key is a set of GGSW ciphertexts $\{\mathrm{GGSW}_{\mathbf{S}'}(s_i)\}_{i=1}^n$ with the decomposition base B_{PBS} and level ℓ_{PBS}. Since the GGSW ciphertexts are fresh, the bootstrapping key can be compressed into seeded GGSW ciphertexts, resulting in the size of $\ell_{\mathsf{PBS}}(k+1)nN \log q$ bits. Let $\mathbf{s}' = (s'_1, \ldots, s'_{kN}) \in \mathbb{B}^{kN}$ be the LWE secret key induced from $\mathbf{S}'$. The keyswitching key is a set of LWE ciphertexts $\{\mathrm{LWE}_{\mathbf{s}}(s'_i \cdot q/B^j_{\mathsf{KS}})\}_{(i,j)\in[kN]\times[\ell_{\mathsf{KS}}]}$ where B_{KS} and ℓ_{KS} are keyswitching decomposition base and level, respectively. As a set of seeded LWE ciphertexts, the keyswitching key is of size $\ell_{\mathsf{KS}}kN \log q$ bits. Table 7 summarizes the size of the evaluation keys for the GenPBS operation.

	Bootstrapping Key	Keyswitching Key
Normal	$\ell_{\mathsf{PBS}}(k+1)^2 nN \log q$	$\ell_{\mathsf{KS}}k(n+1)N \log q$
Seeded	$\ell_{\mathsf{PBS}}(k+1)nN \log q$	$\ell_{\mathsf{KS}}kN \log q$

Table 7: Size of the GenPBS evaluation keys in bits. The size of seeds or auxiliary information is omitted.

GLWEtoGGSW Keysize. For the GLWEtoGGSW conversion proposed in [19], two types of evaluation keys are needed; one is the GLWEtoGLWE keyswitching keys, and the other is the GGSW ciphertext $\mathrm{GGSW}_{\mathbf{S}}(-\mathbf{S})$. The GLWEtoGGSW conversion rotates the coefficients in a GLWE ciphertext by switching a GLWE ciphertext under the secret key $S(X^m)$ to the GLWE ciphertext of the same plaintext under a secret key $S(X)$ for $m = N/2^{i-1} + 1$ where $i = 1, \ldots, \log N$. A GLWEtoGLWE keyswitching key that switches a GLWE ciphertext under $\mathbf{S} = (S_1, \ldots, S_k)$ to the GLWE ciphertext under $\mathbf{S}' = (S'_1, \ldots, S'_k)$[23] is a set of GLWE ciphertexts $\{\mathrm{GLWE}_{\mathbf{S}'}(S_i \cdot q/B^j_{\mathsf{subs}})\}_{(i,j)\in[k]\times[\ell_{\mathsf{subs}}]}$ where B_{subs} and ℓ_{subs} are the decomposition base and level of the GLWEtoGLWE keyswitching in the GLWEtoGGSW conversion. Since the GLWEtoGGSW conversion requires $\log N$ keyswitching keys, the size for them is $\ell_{\mathsf{subs}}kN \log N \log q$ bits as seeded GLWE ciphertexts. The seeded GGSW ciphertext $\mathrm{GGSW}_{\mathbf{S}}(-\mathbf{S})$ with the decomposition base B_{SK} and level ℓ_{SK} is of size $\ell_{\mathsf{SK}}(k+1)N \log q$ bits. Table 8 summarizes the size of the evaluation keys for the GLWEtoGGSW conversion.

Communication Overload for FRAST. Transciphering with FRAST requires two bootstrapping keys: one of the default parameters of size 23.19 MB for FRAST keystream evaluation, and the other of the Kreyvium parameters of size 10.72 MB for the bit extraction and the online phase[24]. The bootstrapping key

[23] In general, $\mathbf{S}'$ may have different GLWE dimension.

[24] There are keyswitching keys used inside the bootstrapping keys and between the bootstrapping keys, but we ignore them since their sizes are of several KBs.

	GLWEtoGLWE Keyswitching Key	$\mathrm{GGSW}_{\mathbf{S}}(-\mathbf{S})$
Normal	$\ell_{\mathsf{subs}}k(k+1)N\log N\log q$	$\ell_{\mathsf{SK}}(k+1)^2N\log q$
Seeded	$\ell_{\mathsf{subs}}kN\log N\log q$	$\ell_{\mathsf{SK}}(k+1)N\log q$

Table 8: Size of the GLWEtoGGSW evaluation keys in bits. The size of seeds or auxiliary information is omitted.

of the default parameters is not taken into account for the communication overload since it is used in the actual usecase after the transciphering. To use double blind rotation technique, FRAST requires additional evaluation keys: the GLWE-toGLWE keyswitching keys of 880 KB, and the GGSW ciphertext $\mathrm{GGSW}_{\mathbf{S}}(-\mathbf{S})$ of 160 KB. The round keys of 4808 bits are packed in $\lceil 4808/N\rceil \cdot \ell_{\mathsf{rk}}$ GLWE ciphertexts of 144 KB, and the remaining round keys of 312 bits are sent to the server in 78 LWE ciphertexts of size 624 B (see Supplementary Material H). Hence, the total communication overload for FRAST is 11.88 MB.

H TFHE Evaluation Methods of FRAST with Error Analysis

In this section, we describe the TFHE evaluation of FRAST using the default parameters described in Section 6, and the error probability of it. For simplicity, we only describe how to evaluate the fixed round functions as evaluating the random round functions using negacyclic S-boxes is simpler.

H.1 TFHE Evaluation of FRAST

The FRAST round function consists of two parts: evaluating multiple S-boxes fed with the first branch added by multiple round keys and evaluating the last S-box of which output is added to the first branch. These two parts are called the expanding and the contracting parts of the round function, respectively. After evaluating FRAST, each bit of the keystream is extracted.

Let $(x_1, \dots, x_\ell)$ be input, $(y_1, \dots, y_\ell)$ be output of the FRAST fixed round function of which S-box is S. We consider the ciphertext modulus of $q = 2^{64}$, the message modulus of $p = 16$, and the scaling factor of $\varDelta = q/p = 2^{60}$.

The input $x_i \in [\![0, p[\![$ is given as an LWE ciphertext scaled by $\varDelta$ without padding, namely, $\mathrm{LWE}(\varDelta \cdot x_i)$, for $i = 1, \dots, \ell$. It can be also regarded as a ciphertext of $2x_i \in [\![0, 2p[\![$ with a scaling factor of $\varDelta/2$ since

$$[\varDelta \cdot x_i]_q = [(\varDelta/2) \cdot (2x_i)]_q.$$

H.1.1 Expanding Part

The expanding part of the FRAST round function is evaluated by the double blind rotation as mentioned in Section 4.2. That said, for the first round, it is possible to evaluate the S-boxes without computing GenPBS on x_1 since the input to the first round is the known constant **ic**. By giving GGSW ciphertexts of the round key bits used in the expanding part of the first round, one can evaluate the expanding part of the first round in a much smaller number of the CMux gates.

For the other rounds, decompose S into $S_{\mathsf{msb\text{-}odd}}$ and $S_{\mathsf{msb\text{-}even}}$. Since $S_{\mathsf{msb\text{-}odd}}$ is negacyclic, it is possible to compute $\mathrm{LWE}(\Delta \cdot S_{\mathsf{msb\text{-}odd}}(x_1 + rk_j))$ for all $j = 2, \ldots, \ell$ in a single GenPBS operation using the double blind rotation. Although the range of $S_{\mathsf{msb\text{-}odd}}$ itself is not $[\![0, p[\![$, one can regard $\mathrm{LWE}(\Delta \cdot S_{\mathsf{msb\text{-}odd}}(x))$ as an LWE ciphertext of $2S_{\mathsf{msb\text{-}odd}}(x) \in [\![0, 2p[\![$ with a scaling factor of $\Delta/2$ as mentioned above.

During the evaluation of $S_{\mathsf{msb\text{-}odd}}$ on the input x_1, it is also possible to extract the MSB bit of x_1 for free using PBSmanyLUT [22].[25] The extracted MSB bit of x_1 is subtracted from x_1, obtaining an LWE ciphertext of $\mathsf{ClearMSB}(x_1) \in [\![0, p/2[\![$ that will be fed into $S_{\mathsf{msb\text{-}even}}$.

To compute $S_{\mathsf{msb\text{-}even}}$ on $\mathsf{ClearMSB}(x_1)$ is possible with a single GenPBS operation using the cleared MSB bit of x_1, while one more padding bit is required for the double blind rotation since $\Delta \cdot (\mathsf{ClearMSB}(x_1) + \mathsf{ClearMSB}(rk_i))$ might exceed $q/2$, filling the padding bit of the ciphertext. Since $S_{\mathsf{msb\text{-}even}}$ is of only 3-bit precision, we address this issue by giving one more padding bit to the ciphertext of $\mathsf{ClearMSB}(x_1)$. Computing $\mathrm{LWE}((\Delta/2) \cdot \mathsf{ClearMSB}(x_1))$ using one more GenPBS operation, one can apply the double blind rotation using GenPBS of 4-bit precision since $\frac{\Delta}{2}(\mathsf{ClearMSB}(x_1) + \mathsf{ClearMSB}(rk_i)) \in [\![q/2^5, q/2[\![$.

It is also possible to refresh the ciphertext of $\mathsf{ClearMSB}(x_1)$ simultaneously during the double blind rotation. Adding the refreshed ciphertext of the MSB of x_1 and $\mathsf{ClearMSB}(x_1)$, which is obtained during adjusting its scaling factor from Δ to $\Delta/2$ to evaluate $S_{\mathsf{msb\text{-}even}}$, one can refresh the first state before adding $S(y_2 + \cdots + y_\ell + rk_1)$, making it possible to use PBSmanyLUT on the input of the first branch.

The number of the CMux gates also can be reduced with a simple tweak: to compute the blind rotation on $S(x_1 + rk_2)$ instead of $S(x_1)$. Then $S(x_1 + rk_j)$ is computed by the second blind rotation by the bits of $rk_j - rk_2$ for $j = 3, \ldots, \ell$. It reduces the number of round key bits to be packed on the GLWE ciphertext, but rk_2 (for each round) should be sent to the server as an LWE ciphertext. Refreshing the state of the first branch also requires additional subtraction by rk_2 since the ciphertext of $x_1 + rk_2$ is refreshed. In summary, evaluating the expanding part of the FRAST round function requires 2 GenPBS operations followed by $7(\ell - 2)$ CMux gates.

[25] The success probability of PBSmanyLUT is sensitive to the parameter and the error contained in the input when N is small, so that it should be used carefully. For FRAST, we have checked that PBSmanyLUT can be used with negligible failure probability.

H.1.2 Contracting Part

The contracting part computes $S(y_2 + \cdots + y_\ell + rk_1)$. If the summation is directly computed from $y_2, \ldots, y_\ell$, the magnitude of the error inside the summation increases with the round. Instead, we use another variable, dubbed crfsum, to manage the noise for the summation.

At first, crfsum is initialized by the trivial encryption of $\sum_{j=2}^{\ell} \mathbf{ic}[j]$. In the expanding part, the output ciphertexts of $S(x_1 + rk_j)$ for $j = 2, \ldots, \ell$ are added to crfsum. Then crfsum added by rk_1 becomes the input of the S-box S in the contracting part, which can be evaluated in 3 GenPBS operations using our WoP-PBS. Using the same idea to refresh the first branch, one can refresh crfsum by one more GenPBS operation. By the help of PBSmanyLUT, both evaluating the S-box and refreshing crfsum can be done using 2 GenPBS operations.

H.1.3 Bit Extraction

A ciphertext containing a FRAST keystream word of 4 bits is decomposed into 4 ciphertexts containing each keystream bit scaled by $q/2$ using the multi-value PBS [18]. Let $x = b_0 + 2b_1 + 2^2 b_2 + 2^3 b_3$ be a keystream word where $b_i \in \{0, 1\}$ for $i = 0, \ldots, 3$. The MSB b_3 can be extracted by the negacyclic function

$$x \mapsto (-1)^{b_3+1} \cdot \frac{q}{2},$$

followed by an addition of $q/2$. The other bits b_i can be extracted by the negacyclic functions

$$x \mapsto b_i \cdot \frac{q}{2}$$

for $i = 0, 1, 2$. All the extraction functions evaluate negacyclic functions on the same input x, so they can be evaluated at the cost of almost one PBS by the multi-value PBS [18].

H.2 Error Analysis

In this section, we analyze the error growth in homomorphic keystream evaluation of FRAST. We use the following conventions as in [22].

- $\mathrm{Var}(E) \le \sigma^2$ for $E \in \mathcal{R}_{q,N}$ if all coefficients of E have variances at most σ^2.
- $\mathrm{Var}(\mathbf{c}) \le \sigma^2$ for a GLWE ciphertext $\mathbf{c}$ if its phase E satisfies $\mathrm{Var}(E) \le \sigma^2$.
- $\mathrm{Var}(\mathbf{C}) \le \sigma^2$ for a GGSW ciphertext $\mathbf{C} = \left(\mathbf{C}^{(i,j)}\right)_{(i,j)\in[k+1]\times[\ell]}$ if all of its GLWE ciphertext components $\mathbf{C}^{(i,j)}$ satisfies $\mathrm{Var}(\mathbf{C}^{(i,j)}) \le \sigma^2$.

Every error associated with those ciphertexts follows a zero-mean distribution in this section. We note that the error analysis in [19] uses a different convention in terms of describing the error, so we properly translated its result to the convention used in this paper with slightly improved bound used in [22]. The formal proof described in [22] is way complicated, so we only give a sketch of how the results in [22] can be applied in the following lemmas.

Lemma 2 (Theorem 4.1 in [19]). *Suppose GLWEtoGLWE keyswitching keys*

$$\mathrm{KS}_m = \left\{ \mathrm{GLWE}_{\mathbf{S}(X^m)} \left(S_i \cdot \frac{q}{B_{\mathsf{subs}}^j} \right) \right\}_{(i,j)\in[k]\times[\ell_{\mathsf{subs}}]}$$

such that $\mathrm{Var}(\mathrm{KS}_m) \le \sigma_{\mathsf{subs}}^2$ *for all* $m = N/2^{i-1}+1$, $i = 1, \ldots, \log N$ *are given. For an input* $\mathbf{c} = \mathrm{GLWE}_{\mathbf{S}}(\sum_{i=0}^{N-1} b_i X^i)$, *Algorithm 3 in [19] outputs a set of ciphertexts* $\{\mathbf{c}_j = \mathrm{GLWE}_{\mathbf{s}}(N b_j)\}_{j=0}^{N-1}$ *with noise variance*

$$\mathrm{Var}(\mathbf{c}_j) \le N^2\,\mathrm{Var}(\mathbf{c}) + \frac{N^2-1}{3} V_{\mathsf{sk}}$$

where

$$V_{\mathsf{sk}} = \frac{kN}{2}\left(\frac{q^2}{12 B_{\mathsf{subs}}^{2\ell_{\mathsf{subs}}}} - \frac{1}{12} \right) + \frac{kN}{16} + kN\ell_{\mathsf{subs}}\sigma_{\mathsf{subs}}^2 \frac{B_{\mathsf{subs}}^2+2}{12}.$$

Proof. The noise increment V_{sk} of the GLWEtoGLWE keyswitching can be estimated by the result of Appendix D in [22] with a slight modification. For each iteration on i in Algorithm 3 in [19], the noise increases by V_{sk} for each GLWEtoGLWE keyswitching. Then one can obtain the above upper bound following the proof of Theorem 4.1 in [19] (considering the difference of the convention as mentioned before). □

Lemma 3 (Theorem 4.2 in [19]). *Let* $\mathbf{A} = \mathrm{GGSW}_{\mathbf{S}}(-\mathbf{S})$ *be a GGSW ciphertext of* $-\mathbf{S}$ *with the base* B_{SK} *and level* ℓ_{SK}. *For inputs* $\mathbf{A}$ *and GLWE ciphertexts* $\{\mathbf{c}_j = \mathrm{GLWE}_{\mathbf{S}}(\sum_{i=0}^{N} \frac{b_i X^i}{N B_{\mathsf{rk}}^{j+1}})\}_{j=1}^{\ell_{\mathsf{rk}}}$, *Algorithm 4 in [19] outputs GGSW ciphertexts* $\mathbf{C}_i = \mathrm{GGSW}_{\mathbf{S}}(b_i)$ *with the base* B_{rk} *and level* ℓ_{rk} *of which noise variance is given by*

$$\begin{aligned}\mathrm{Var}(\mathbf{C}_i) \le N^2\,\mathrm{Var}(\mathbf{c}) + \frac{N^2-1}{3} V_{\mathsf{sk}} + \ell_{\mathsf{SK}}(k+1)N\frac{B_{\mathsf{SK}}^2+2}{12}\,\mathrm{Var}(\mathbf{A}) \\ + \frac{q^2 - B_{\mathsf{SK}}^{2\ell_{\mathsf{SK}}}}{12 B_{\mathsf{SK}}^{2\ell_{\mathsf{SK}}}}\left(1+\frac{kN}{2}\right) + \frac{kN}{8} + \frac{1}{4}\left(1-\frac{kN}{2}\right)^2 \end{aligned} \qquad (14)$$

for all $i = 0, \ldots, N-1$.

Proof. One can obtain the above bound following the proof of Theorem 4.2 in [19] directly (considering the difference of the convention as mentioned before), except that the GGSW ciphertexts $\mathbf{A}$ and $\mathbf{C}_i$ have different decomposition base and level. □

We denote the upper bound in (14) by σ_{dbr}^2 for the rest of this section.

Lemma 4 (Theorem 4 in [22]). *The noise variance of the GenPBS output is*

$$\begin{aligned}\mathrm{Var}(\mathsf{PBS}) = n\ell(k+1)N\frac{B_{\mathsf{PBS}}^2+2}{12}{\sigma_{\mathsf{PBS}}}^2 \\ + n\frac{q^2 - B_{\mathsf{PBS}}^{2\ell_{\mathsf{PBS}}}}{24 B_{\mathsf{PBS}}^{2\ell_{\mathsf{PBS}}}}\left(1+\frac{kN}{2}\right) + \frac{nkN}{32} + \frac{n}{16}\left(1-\frac{kN}{2}\right)^2\end{aligned}$$

where σ_{PBS} is the noise variance of the bootstrapping key, and B_{PBS} and ℓ_{PBS} are the base and level of GenPBS, respectively.

Lemma 5. *Let $\mathbf{C}_i$ be a GGSW ciphertext of noise variance at most σ^2 of the base B and the level of ℓ for $i = 1, \ldots, t$. Let $\mathbf{c}$ be a GLWE ciphertext of noise variance at most σ^2_{prev}. When the nested external products on $\mathbf{c}$ by $\mathbf{c}' = \mathbf{C}_t \boxdot \cdots (\mathbf{C}_2 \boxdot (\mathbf{C}_1 \boxdot c))$ is well-defined, the noise variance of $\mathbf{c}'$ is bounded by*

$$\begin{aligned}Var(\mathbf{c}') \leq \sigma^2_{\mathsf{prev}} + t^2\ell(k+1)N\frac{B^2+2}{12}\sigma^2 \\ + t\left(\frac{q^2 - B^{2\ell}}{12B^{2\ell}}\left(1+\frac{kN}{2}\right) + \frac{kN}{8} + \frac{1}{4}\left(1-\frac{kN}{2}\right)^2\right).\end{aligned}$$

Proof. Following the analysis of Appendix B in [22] similarly with a partial Assumption 3.11(Independence heuristic) in [20], we may claim the term appearing in Step (1) of Appendix B has at most quadratic growth over the iteration of external products, and the term appearing in Step (2) has a linear growth because of their relative independence. □

Lemma 6. *In the double blind rotation, t nested CMux gates by GGSW ciphertexts of noise variance at most σ_{dbr} after the common blind rotation increases the output noise variance by*

$$\begin{aligned}V_{\mathsf{DBR},t} \leq t^2\ell_{\mathsf{rk}}(k+1)N\frac{B^2_{\mathsf{rk}}+2}{12}\sigma^2_{\mathsf{dbr}} \\ + t\left(\frac{q^2 - B^{2\ell_{\mathsf{rk}}}_{\mathsf{rk}}}{12B^{2\ell_{\mathsf{rk}}}_{\mathsf{rk}}}\left(1+\frac{kN}{2}\right) + \frac{kN}{8} + \frac{1}{4}\left(1-\frac{kN}{2}\right)^2\right).\end{aligned}$$

where B_{rk} and ℓ_{rk} are the base and level of the GGSW ciphertext, respectively.

Proof. The noise increment by the CMux gate can be estimated by the result of Appendix B in [22] along with Lemma 5. The above upper bound is for the worst-case such that the distribution of the plaintext of the GGSW ciphertext is unknown over $\mathbb{B}$, considering the dependency of the round key bits in FRAST obtained from its master key. □

Using the above lemmas, one can upper bound the noise variance of the S-box outputs obtained by the double blind rotation as described in Section H.1.1. After decomposing the S-box S into $S_{\mathsf{msb\text{-}odd}}$ and $S_{\mathsf{msb\text{-}even}}$, one computes the common blind rotation for $S_{\mathsf{msb\text{-}odd}}$ and $S_{\mathsf{msb\text{-}even}}$ on the input $x_1 + rk_2$ and $\mathsf{ClearMSB}(x_1 + rk_2)$, respectively. Then, 4 (resp. 3) CMux gates follow using GGSW ciphertexts of each bit of $rk_j - rk_2$ (resp. $\mathsf{ClearMSB}(rk_j - rk_2)$) for $j = 3, \ldots, \ell$. From Lemma 6, the noise variance of the output ciphertext of $S_{\mathsf{msb\text{-}odd}}(x_1 + rk_j)$ (resp. $S_{\mathsf{msb\text{-}even}}(\mathsf{ClearMSB}(x_1 + rk_j)))$ by the double blind rotation is given as Var(PBS) increased by $V_{\mathsf{DBR},4}$ (reps. $V_{\mathsf{DBR},3}$).

In the evaluation of the FRAST round function, the noisiest ciphertext is one for the input to the S-box in the contracting part of the fixed round function, which is named crfsum in Section H.1.1. Considering the output noise variance of the resulting ciphertexts of the blind rotation, one can upper bound the noise increment of crfsum in each round by

$$2(\ell-1)^2\,\mathrm{Var}(\mathsf{PBS}) + (\ell-2)(V_{\mathsf{DBR},3} + V_{\mathsf{DBR},4})$$

assuming that there is no correlation between the noises resulting from the common blind rotation and the GGSW ciphertexts of the round key bits, while there is clear dependence on the noises from the common blind rotation.

Considering additional noise increments such as the initial noise in refreshed crfsum, the noise in rk_1 and the keyswitching noise before the GenPBS operation[26], all of which do not affect the failure probability significantly, we obtain noise of standard deviation $2^{54.17}$ for crfsum, small enough compared to the scaling factor $\varDelta = 2^{60}$. We observed the upper bound of noise coincides empirically with our result as Figure 10 shows the result of noise measurement to crfsum in 1000 evaluations of FRAST for each round. The failure probability of the GenPBS operation on the input of crfsum computed by Theorem 3 in [22] is negligible, and even PBSmanyLUT with $\vartheta = 1$, i.e., computing two functions on the same input in a single GenPBS call, can be used with failure probability less than 2^{-80}.

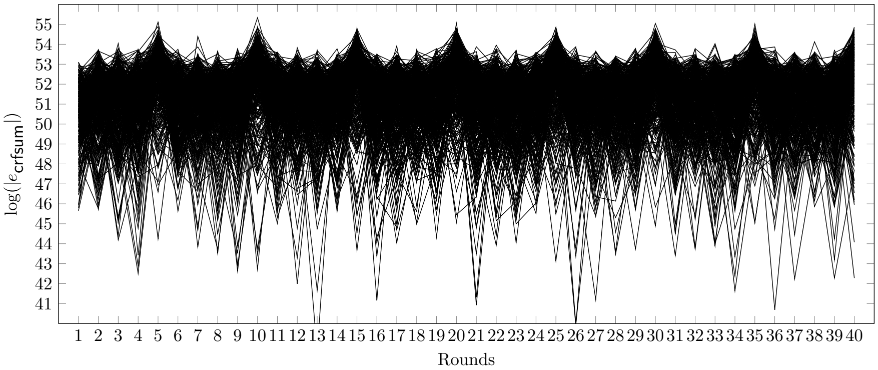


Fig. 10: The magnitude of error of crfsum in each round of FRAST evaluation. The experiment is performed 1000 times.

[26] It can be computed by Theorem 2 in [22].