FASTA – a stream cipher for fast FHE evaluation

Carlos Cid[1,2][0000−0001−5761−8694], John Petter Indrøy[2], and Håvard Raddum[2][0000−0001−9779−5986]

[1] Royal Holloway University of London, Egham, United Kingdom
carlos.cid@rhul.ac.uk
[2] Simula UiB, Bergen, Norway
{johnpetter,haavardr}@simula.no

Abstract. In this paper we propose FASTA, a stream cipher design optimised for implementation over popular fully homomorphic encryption schemes. A number of symmetric encryption ciphers have been recently proposed for FHE applications, e.g. the block cipher LowMC, and the stream ciphers Rasta (and variants), FLIP and Kreyvium. The main design criterion employed in these ciphers has typically been to minimise the multiplicative complexity of the algorithm. However, other aspects affecting their efficient evaluation over common FHE libraries are often overlooked, compromising their real-world performance. Whilst FASTA may also be considered as a variant of Rasta, it has its parameters and linear layer especially chosen to allow efficient implementation over the BGV scheme, particularly as implemented in the HElib library. This results in a speedup by a factor of 25 compared to the most efficient publicly available implementation of Rasta. FASTA's target is BGV, as implemented in HElib. However the design ideas introduced in the cipher could also be potentially employed to achieve improvements in the homomorphic evaluation in other popular FHE schemes/libraries. We do consider such alternatives in this paper (e.g. BFV and BGVrns, as implemented in SEAL and PALISADE), but argue that, unlike BGV in HElib, it is more challenging to make use of their parallelism in a Rasta-like stream cipher design.

Keywords: Stream Ciphers · Homomorphic Encryption · Hybrid Encryption

1 Introduction

Fully homomorphic encryption (FHE) is a relatively new and active research area in cryptography. FHE schemes allow arbitrary operations to be performed on ciphertexts, to produce some encrypted result, which when decrypted results in data that would be obtained if we had decrypted the ciphertexts first and then performed the operations on the plaintexts.

FHE opens up for new and exciting secure applications, in particular in cloud computing. The party doing the operations on the ciphertexts does not need to

have the decryption key. One can therefore upload FHE-encrypted ciphertexts to the cloud and have the cloud provider perform the necessary operations on the ciphertexts. Since the cloud does not need the decryption key, there is no need to place any trust in the cloud provider. This gives a higher level of security as the cloud provider does not have the ability to read the plaintext information.

The main drawback of FHE is that it is very computationally demanding. Since Gentry demonstrated the first FHE scheme [Gen09] in 2009 many improvements in efficiency have been made [CIM16,DM15,CHK20], but the most useful applications still struggle with being practical. This impracticality comes not least because clients of a cloud need to perform FHE encryptions themselves. One notices however that the computing power of a cloud is much higher than that of a typical client, so research has gone into finding ways to transfer most of the burden of doing FHE encryptions from the clients to the cloud.

A solution for achieving this goal is to let the client encrypt its data using a symmetric cipher, which is computationally very cheap, and upload the symmetrically encrypted ciphertexts to the cloud. The cloud also receives the key used for the symmetric encryption, but only as a ciphertext encrypted under the FHE scheme. The cloud is then in a position to *homomorphically* remove the symmetric encryption and end up with the FHE encryption of the client's data.

A number of symmetric ciphers designed for use together with FHE have been proposed, e.g. the block cipher LowMC [ARS+15], and the stream ciphers Kreyvium [CCF+16], FLIP [MJSC16], and Rasta [DEG+18] (and variants [HL20,HKC+20,DGH+21b]). Their main design criterion has been to minimise the multiplicative complexity of the algorithms since homomorphic multiplications are the most expensive operations in FHE. However, as a rule they have mostly overlooked an important aspect for their application target: how suitable they are for their homomorphic evaluation over existing FHE schemes, as implemented in the main libraries. For example, the HElib and PALISADE libraries [HS20,PAL] implement the BGV scheme [BGV12], which offers a good degree of parallelism by utilising *slots* in BGV ciphertexts. The BFV scheme, implemented in PALISADE and SEAL [SEA20], also offer the same kind of parallelism. Since these are some of the most popular FHE implementations, one may argue that a symmetric encryption design should – in addition to minimising multiplicative complexity – also select its components and parameters to take advantage of the libraries' features to allow their efficient homomorphic evaluation.

In this paper we propose Fasta, a stream cipher design optimised for implementation over HElib. Fasta may be considered as a variant of Rasta, but has its parameters and linear layers especially chosen to allow efficient implementation over the BGV scheme. The selected parameters utilise the parallelism offered by BGV, where the slots in ciphertexts are packed to achieve full parallelisation when evaluating the non-linear layer. However the packing is inefficient when the linear layer consists of random matrices (as with Rasta). Thus Fasta also features a new BGV-friendly linear layer. These changes result in Fasta running

more than 6 times faster than a corresponding (modified) Rasta instance, and 25 times faster than the original Rasta, when evaluated homomorphically.

Whilst FASTA's target is BGV, as implemented in the HElib library, we also look into the BFV scheme implemented in PALISADE and SEAL, and the variant of BGV called BGVrns that is implemented in PALISADE. We consider the implementation features in these libraries and explain why it is more challenging to make good use of their parallelism in Rasta-like stream ciphers.

The paper is organised as follows. In Section 2 we give an overview of the main concepts and schemes discussed in the paper. Section 3 focuses on the design of symmetric key linear layers for efficient FHE evaluation. We specify the FASTA stream cipher in Section 4, and provide a security analysis in Section 5. We describe the homomorphic implementation of FASTA in Section 6, and close with our conclusions in Section 7.

2 Preliminaries

In this section we briefly recall a main use case for using symmetric ciphers with homomorphic encryption schemes. We also review the Rasta stream cipher and the BGV scheme, in particular how the latter is implemented in popular FHE libraries.

2.1 FHE Hybrid Encryption: Combining Symmetric Ciphers with Fully Homomorphic Encryption

The concept of Fully Homomorphic Encryption (FHE) was first described in [RAD78]. However no actual FHE schemes were found before Gentry proposed a construction in 2009 [Gen09]. Since then much work has been invested in this field, not least because FHE gives strong solutions to privacy problems related to cloud computing. The problem that FHE faces today concerns computational efficiency. Significant improvements have been made in recent years, but efficiency is still a bottleneck for deploying practical and useful FHE applications.

One approach to address the efficiency issue is to combine FHE schemes with symmetric ciphers as shown in Figure 1. This is often referred to as *FHE hybrid encryption*. The idea is that clients in a cloud system, who typically have much less computational power than the cloud provider, rather than homomorphically encrypting a (potentially large) plaintext P, will instead encrypt P using a symmetric cipher $\mathcal{E}$ under a secret key K, and then only homomorphically encrypt K under the FHE scheme HE using a public key pk. Both the ciphertext $C = \mathcal{E}_K(P)$ and the FHE-encrypted key K^* are uploaded to the cloud.[3] The cloud will now encrypt the bits in C using HE under the public key pk, and homomorphically run the decryption circuit of $\mathcal{E}$ on the inputs $C^* = HE(C, pk)$

[3] To avoid confusion between symmetric and FHE ciphertexts, we will normally use an asterisk "*" as a superscript on any literal denoting a FHE ciphertext.

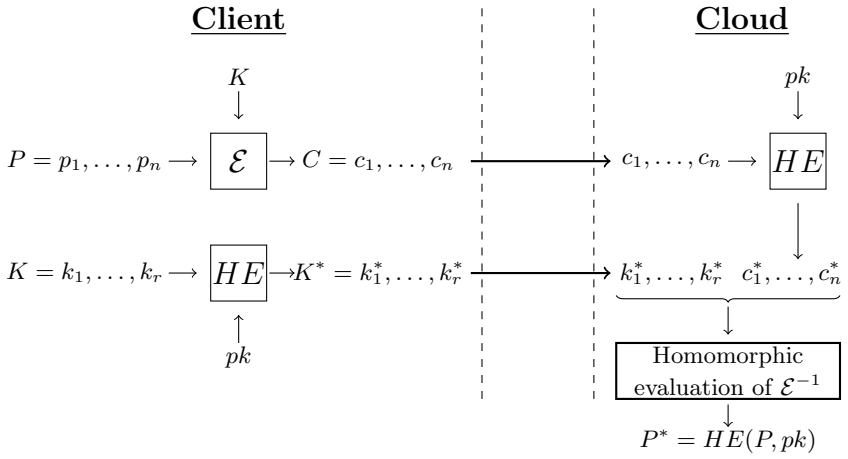


Fig. 1: FHE hybrid encryption: the client only needs to encrypt the key K with an FHE scheme HE; the plaintext P is encrypted with symmetric algorithm $\mathcal{E}$. The cloud gets the bits of K encrypted under HE, it encrypts the ciphertext bits c_i with HE, and homomorphically evaluates the decryption circuit of $\mathcal{E}$ to obtain $HE(P, pk)$.

and $K^* = HE(K, pk)$. The homomorphic properties of HE ensure that the output from doing this is $HE(P, pk)$.[4] In other words, the effect of the symmetric cipher can be removed, and the cloud is now left with a pure FHE encryption of P, which may then be used for further processing. The benefit of this construction is that the client side only needs to encrypt K using HE – in fact it needs not be the same device that encrypts the plaintext P with $\mathcal{E}$, and K with HE. All other homomorphic encryptions and evaluations are done by the cloud.

The basic homomorphic operations performed in a circuit are additions and multiplications, corresponding to the bit-wise XOR and AND operations when the plaintext space is $\mathbb{F}_2$. Both of these operations have a cost in terms of the growth of *noise*, and multiplication is by far the most expensive. Thus, to support such FHE hybrid encryption construction, there has been much research effort in designing symmetric ciphers that minimise the multiplicative complexity – the number of bit-wise AND-gates, both in the total number and in a critical path (i.e. the AND-depth) – of their decryption circuit. Examples include LowMC[ARS+15], FLIP [MJSC16], Kreyvium [CCF+16], and Rasta and its variants Dasta, Masta and Pasta [DEG+18,HL20,HKC+20,DGH+21b].

2.2 The Rasta stream cipher

Rasta is a family of stream ciphers proposed by Dobraunig *et al.* [DEG+18]. The target application for the ciphers is the use as a component in secure computa-

[4] Strictly speaking, the result will be in fact a ciphertext which will decrypt to P under the FHE private key sk.

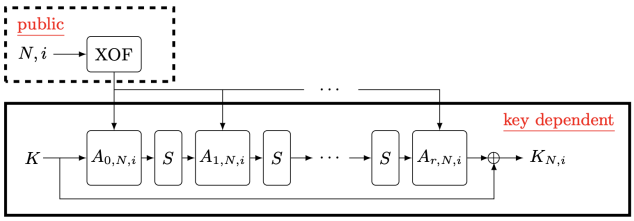


Fig. 2: The r-round Rasta keystream generator construction (from [DEG+18]).

tion constructions based on MPC and FHE schemes, particularly the latter. In these applications, symmetric key algorithm designs will seek to minimise multiplications as much as possible. In the Rasta construction, the designers aimed to minimise two multiplicative metrics of interest: AND-depth and ANDs per encrypted bit. Rasta uses a cryptographic permutation based on a public and fixed substitution layer, and variable affine layers (which are derived from public information), iterated for d rounds. The construction achieves AND-depth d, while requiring only d ANDs per encrypted bit.

In more detail, the Rasta keystream generator is based on a n-bit permutation featuring the $A(SA)^d$ structure, where S is the χ-transformation (prominently also used in Keccak [BDPA11]), and the j^{th}-round affine layers $A_{j,N,i}$ are generated pseudorandomly based on a nonce N and a counter i. To produce the keystream, it applies the permutation in feed-forward mode, with the n-bit secret key K as input. Figure 2 shows a diagrammatic representation of the Rasta keystream generator.

The generation procedure for the affine layers $A_{j,N,i}$ results in pseudorandomly generated $n \times n$ invertible binary matrices and n-bit round constants, which since they are based on unique (N, i), are unlikely to be re-used during encryption under the same key. To ensure S is invertible, we require n to be odd. If the permutation has d rounds, it is straightforward to show that the Rasta construction achieves AND-depth d and requires d ANDs per encrypted bit.

In [DEG+18], the authors suggest several parameter sets for 80-, 128- and 256-bit security. For example, Rasta with a 6-round permutation with block/key size of 219 bits should provide 80 bits of security. Same for a 4-round permutation with 327-bit block/key. On the other hand, Rasta based on a 6-round permutation with block/key size 351 bits is expected to provide 128 bits of security (see Table 1 of [DEG+18] for other proposed parameters). In general, the authors suggest the number of rounds to be between 4 and 6, while the key size will typically be at least three times larger than the security level. However they also propose a more "aggressive" version of the cipher (Agrasta), for which the block size coincides with the security level (plus one, to ensure n is odd). For example, Agrasta based on an 81-bit, 4-round permutation, claims 80 bits of security.

In order to derive and justify Rasta's parameter choices, the authors provide a detailed security analysis of the construction in [DEG+18]. To our best knowledge, the only other publicly available cryptanalysis of Rasta[5] (and variants) is the recent work [LSMI21], proposing algebraic attacks that contradict some of the security claims in [DEG+18].

Rasta's designers also discuss a few areas for future work, in particular how to improve the cipher's affine layer. They state in [DEG+18] that "[n]ew ideas for linear-layer design are needed which impose structure in one way or another which on one hand allows for significantly more efficient implementations while at the same time still resist attacks and allows for arguments against such attacks." A variant of Rasta, called Dasta [HL20] was later proposed, considering a particular efficiency aspect: it features a more efficient generation procedure for the linear layer, which does not make use of a XOF algorithm. In this paper we consider another implementation efficiency aspect: the evaluation of Rasta-like ciphers over popular FHE schemes and libraries.

We note that, in [DEG+18], the designers did describe a few experiments for the main use case for Rasta – namely, the homomorphic evaluation of the cipher in a hybrid symmetric/FHE construction. However these experiments, using BGV as implemented in HElib, appeared to have been carried out mainly to "validate" the Rasta design approach, as well as a means to compare it with other prominent ciphers, e.g. FLIP, Kreyvium and LowMC. In particular, there appeared to be no efforts to take advantage of features of BGV/HElib in a more efficient implementation, which in turn might have fed into more efficient design choices for the cipher (beyond simply minimising AND-depth and AND per bit). More recent variants of Rasta [HKC+20,DGH+21b] do take into account FHE schemes' features in their design, and come accompanied by comprehensive experiments. However they feature more distinctive structures, e.g. they are defined over fields of prime characteristic $p > 2$. In contrast, in this paper we propose Fasta as a closer variant of Rasta, also defined over the binary field, in which however we carefully consider the features of BGV in the design of its keystream generator.

2.3 The BGV scheme

The BGV homomorphic encryption scheme [BGV12] was proposed by Brakerski, Gentry and Vaikuntanathan in 2012 and is implemented in the HElib and PALISADE libraries. BGV is a levelled FHE scheme, which means that the multiplicative depth of the circuit one wants to evaluate must be known at the time the parameters of the cipher are chosen.

The starting point for the BGV scheme is the m-th cyclotomic polynomial over the integers $\Phi_m(X)$. Plaintexts in BGV can be seen as elements of the quotient ring $\mathbb{Z}_{p^r}[X]/(\overline{\Phi}_m(X))$, where p is a prime and $\overline{\Phi}_m(X)$ is the image

[5] The designers also mention in [DEG+18] the technical report "Algebraic cryptanalysis of RASTA", by Bile, Perret and Faugère. However we were unable to publicly locate this work.

of $\Phi_m(X)$ in $\mathbb{Z}_{p^r}[X]$. In this paper we are only interested in encrypting bits as plaintext, i.e. $p = 2$ and $r = 1$, and so in fact our plaintexts can be seen as polynomials over $\mathbb{F}_2$ of degree less than $\phi(m)$, where $\phi(\cdot)$ is Euler's totient function. A very useful feature of BGV is that one ciphertext may encrypt several plaintext bits. The notion is that one ciphertext contains multiple *slots*. The number of slots in a ciphertext is denoted by s, which is understood differently in HElib and PALISADE. In HElib the number of slots is given as $s = \phi(m)/d$, where d is the multiplicative order of the size of the plaintext space (in our case, 2) modulo m. In PALISADE the number of slots is given as $s = \phi(m)$. In both cases we use the notation

$$c^* = \{(b_1, b_2, \ldots, b_s)\}$$

to indicate that the ciphertext c^* encrypts the plaintext bits $b_1, \ldots, b_s$.

The homomorphic properties of BGV apply slot-wise. If $c_a^* = \{(a_1, \ldots, a_s)\}$ and $c_b^* = \{(b_1, \ldots, b_s)\}$ are two ciphertexts, then

$$\begin{aligned} c_a^* + c_b^* &= \{(a_1 \oplus b_1, \ldots, a_s \oplus b_s)\}, \\ c_a^* \times c_b^* &= \{(a_1 \cdot b_1, \ldots, a_s \cdot b_s)\}, \end{aligned}$$

where $\oplus$ and $\cdot$ denote the bit-wise XOR and AND operations, respectively.

BGV in HElib. If we have $\phi(m) = s \cdot d$ as above, it follows from the structure of the ring $\mathbb{F}_2[X]/(\overline{\Phi}_m(X))$ that the plaintext space in HElib can be understood to be instead in $\mathbb{F}_{2^d}$, and multiplications and additions work homomorphically in this field (see [HS20]). As $\mathbb{F}_2 \subset \mathbb{F}_{2^d}$, we can use HElib for our purpose, and ciphertexts will encrypt s plaintext bits.

HElib contains functions to manipulate the slots in a ciphertext, and two of these will be important to us. The first is $\mathsf{mul}(c^*, M)$, where c^* is a ciphertext and M is a binary $s \times s$ matrix. The function[6] returns a ciphertext that encrypts the slots in c^* multiplied with M, and so when $c^* = \{(b_1, \ldots, b_s)\}$, we have

$$\mathsf{mul}(c^*, M) = \{((b_1, \ldots, b_s) \cdot M)\}.$$

The second function we would like to highlight is $\mathsf{rotate}(c^*, a)$. This function returns a ciphertext that encrypts the slots of c^* cyclically rotated by a positions to the right. We also use the notation $(c^* >> a)$ for the rotate operation, so for $c^* = \{(b_1, \ldots, b_s)\}$ we have

$$\mathsf{rotate}(c^*, a) = (c^* >> a) = \{(b_{s-a+1}, \ldots, b_s, b_1, \ldots, b_{s-a})\}.$$

We note that both rotate and additions of ciphertexts are computationally very cheap to do, while mul is not.

[6] The mul function was optimised in HElib in March 2018, the earlier name for the same function was matMul [HS18].

BGV in PALISADE. PALISADE implements the BGV scheme using residue number systems, and works in a different fashion from HElib. This particular scheme is denoted by BGVrns.[7] As noted above, the number of slots in PALISADE is $s = \phi(m)$, and will therefore always be an even number. In PALISADE v.1.11.4 (the latest version at the time of writing [PRRC21]), the plaintext space of BGVrns can only be integers modulo a chosen plaintext modulus p. Addition and multiplication in the slots will be performed as integer additions and multiplications modulo p. As we are only interested in doing operations in $\mathbb{F}_2$ and not in any extension field, this is again sufficient for our purpose. In BGVrns the plaintext modulus needs to be odd, but by selecting p to be high enough that our computation never reaches it, the computations will simply be done over the integers. After decryption we then only need to reduce the plaintext returned by PALISADE modulo 2 to get the desired result.

PALISADE does not yet implement a function similar to HElib's mul. It does however have a function that cyclically rotates a ciphertext by a given number of positions, called evalAtIndex. Like HElib, both evalAtIndex and additions are computationally cheap to do in BGVrns, but the number of slots in PALISADE's BGVrns is much higher.

3 Linear layers in symmetric ciphers for FHE hybrid encryption

The purpose of the linear layer in a symmetric cipher is to provide "diffusion". The concept of diffusion is often not precisely formalised, but intuitively we would like a linear layer to provide an *avalanche effect*, i.e. that any single bit of the cipher state at a particular point of the encryption process quickly influences as many bits in the cipher state as possible after a few rounds. Deploying linear layers with good diffusion – together with good non-linear layers – in iterated constructions should ensure that, for the entire cipher, the output bits are described via complex expressions in all input bits.

The notion of *optimal diffusion* for symmetric encryption linear layers was introduced in [Dae95,RDP+96], together with a metric to quantify the diffusion of a linear layer L. The *branch number* of L is defined as the minimum of the sums of the weights of inputs and corresponding outputs of L. For matrices of dimension n over $\mathbb{F}_{2^r}$ $(r > 1)$, it was shown how maximum distance separable (MDS) codes of length $2n$ and dimension n can be used to construct invertible linear transformations providing optimal diffusion.

In this work we are interested in large, invertible linear transformations over $\mathbb{F}_2$, which can offer good diffusion. Given our parameters, the use of the MDS construction is not possible, and measuring the branch number of individual matrices seems infeasible. Similar to the approach in [ARS+15,DEG+18], we will instead define a family of linear transformations which we will argue offer good diffusion properties. FASTA's iterated construction will then use linear layers that

[7] See [HPS18] for a discussion on the very similar BFVrns scheme.

are pseudorandomly generated from this family. We claim that the construction should then provide strong diffusion after just a few rounds.

Most existing work on quantifying diffusion have focused on features of one particular linear transformation used multiple times in a cipher. Our case is different: we will make use of a *family* of linear transformations, from which we will draw transformations to be used only once during encryption. We therefore introduce the notion of "balanced diffusion" which we will use in our construction.

Definition 1. *Let $\mathcal{L}$ be a family of invertible $n \times n$ matrices over $\mathbb{F}_2$, where $|\mathcal{L}|$ is a large even number. Let $\mathbf{e}_0, \ldots, \mathbf{e}_{n-1}$ be the canonical basis of $(\mathbb{F}_2)^n$. Then we say that $\mathcal{L}$ offers balanced diffusion if, for all $0 \leq i, j \leq n-1$, we have*

$$\Pr_{L \in \mathcal{L}}[\langle \mathbf{e}_i L, \mathbf{e}_j \rangle = 1] = 1/2.$$

Intuitively it means that if L is a member of a family of matrices offering balanced diffusion, then if $\mathbf{w} = \mathbf{v} \cdot L$, we expect that every bit of $\mathbf{v}$ influences each bit of $\mathbf{w}$ with probability $1/2$. In cryptographic applications, we expect that the iteration of randomly generated members of $\mathcal{L}$ should maximise the diffusion of the entire construction.

Some designers of FHE-friendly symmetric ciphers, e.g. in [ARS+15,DEG+18], have adopted a similar approach, using $\mathcal{L} = GL(n, \mathbb{F}_2)$ the family of all invertible $n \times n$ binary matrices. The ciphers' round linear transformations are then randomly generated from $\mathcal{L}$. This seems in principle to make sense: designers mainly focused on minimising the number of AND gates and the AND-depth of the decryption circuit, under the argument that linear operations on FHE ciphertexts are almost for free compared to multiplications. Moreover, with no particular structure in the linear layer that a cryptanalyst can exploit in an attack, this approach also simplifies the arguments in the security analysis. However this approach seems also to indicate that little attention was paid to how the structure of the linear layer may affect the performance of the ciphers' homomorphic evaluation in practice.

While it is true that addition of homomorphic ciphertexts is cheap compared to multiplication, a tacit assumption is that ciphertexts only encrypt a single bit each. As discussed in Section 2.3, popular FHE libraries have the ability to pack multiple plaintext bits into a single FHE ciphertext, and operate on all bits encrypted into each ciphertext in parallel. Packing the full state of a symmetric cipher into a few, or perhaps only one, FHE ciphertexts can give big speed-ups when processing the non-linear layer of a symmetric cipher. For example, an S-box layer of LowMC that covers $3/4$ of the state can be processed with only three FHE multiplications, while the χ transformation used in Rasta (Section 2.2) can be performed with only one homomorphic multiplication.

However, when packing the state of a symmetric cipher into few FHE ciphertexts, the additions carried out in a linear layer will now fall into two categories:

1. additions of elements in the same slot positions from two FHE ciphertexts;

2. additions of elements from two FHE ciphertexts in different slots, or addition of elements from different slots inside a single FHE ciphertext.

The first type is in fact the addition of two FHE ciphertexts, and is therefore quick and easy to perform. The second type is however slower and more involved, as it mixes elements inside a single FHE ciphertext, or moves elements inside a ciphertext to make them line up in the same slot. Type 2 additions are thus not homomorphic additions per se. For a randomly generated linear layer, most additions will be of type 2; that in turn will outweigh much of the gains that packed ciphertexts give in the non-linear layer. A natural question is then to investigate whether we can find another family of linear transformations, which only uses additions of type 1, but is still expected to offer balanced diffusion.

We now describe the design of a family of linear layers that only use rotations and additions of type 1, which we employ in Fasta. Of course, linear transformations drawn from this family are no longer *random*, and some structure may be found in them. Nevertheless, we aim to construct linear transformations that are still expected to provide balanced diffusion, as per Definition 1, and which in respect to the diffusion at least, behave as randomly generated binary matrices.

3.1 Rotation-based linear layers

In Fasta, we follow the principle introduced in Rasta to (pseudorandomly) draw linear transformations from a large family $\mathcal{L}$, to be used only once in a particular instantiation of the cipher. Below we describe a general method for constructing a family of FHE-friendly linear transformations, with the aim of providing balanced diffusion. In Section 4 we use this method to construct the specific class of linear transformations used in Fasta. In the following explanation we use the notation $v[i]$ to indicate bit number i in a bit-string v, and $v[i,j], i < j$ to indicate the sequence of bits from position i up to and including position j in v.

The linear transformations we produce are based on *column parity mixers* [SD18]. Let the cipher state consist of bs bits, split into b words $w_0, \ldots, w_{b-1}$ of s bits each. A column parity mixer works by first computing $u = w_0 \oplus \ldots \oplus w_{b-1}$ and then applying a simple linear transformation Θ to the sum to compute $v = \Theta(u)$. The word v is added back onto the input words to form the output words w'_i of the column parity mixer, as $w'_i = w_i + v$. See Figure 3 for a schematic description. In the following we also refer to one application of the column parity mixer as an *iteration*.

We are concerned with constructing a class of linear transformations $\mathcal{L}$ of dimension bs over $\mathbb{F}_2$ that provides close to balanced diffusion. Let $x_0, \ldots, x_{b-1}$ be the input words to any $L \in \mathcal{L}$, and $y_0, \ldots, y_{b-1}$ be the output words. For any $0 \leq i, k < b$ and $0 \leq j, l < s$, we want $x_i[j]$ to appear in the linear expression for $y_k[l]$ for approximately half of the linear transformations in $\mathcal{L}$. As we will only use rotations of words and the column parity mixer construction, we can without loss of generality focus on $x_0[0]$ and ensure this bit will appear in approximately half of the linear combinations giving output bits $y_k[l]$. We say that a bit in any

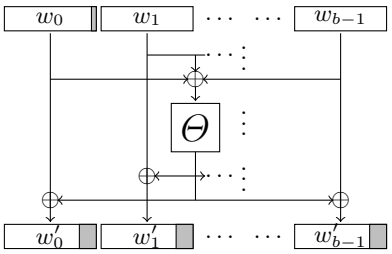


Fig. 3: One iteration of a column parity mixer used to construct rotation-based linear layers (Step 1). Gray areas indicate affected bits.

word w_i during the computation of the linear transformation is *affected* if it has a non-zero probability of depending on $x_0[0]$. We propose the following general strategy for constructing a family of rotation-based linear layers:

1. Define a column parity mixer based on a transformation Θ that uses rotations of low amounts compared to s, in such a way that all bits in a small neighbourhood of $w_0[0]$ will be affected in all words output from the column parity mixer (see Figure 6 in Section 4 for an example of such Θ, as used in Fasta).
2. Rotate the words w_i between applications of the column parity mixer such that the affected parts are spread to larger portions of the cipher state.
3. Iterate applications of column parity mixers interleaved with word rotations as many times as required until the whole cipher state is affected.

We note that if b is even the column parity mixer (step 1 above) is an involution; if b is odd, the column parity mixer operation is invertible iff $(\Theta + \mathbf{I})$ is invertible [SD18]. Moreover, let $w'_0, \ldots, w'_{b-1}$ be the cipher state after the application of the column parity mixer. Then Θ should be designed such that $w'_i[0, a-1]$ is affected for all $0 \leq i < b$ and some value of a relatively small compared to s. This is shown in Figure 3.

After the first iteration the a least significant bits of each output word w'_i will be affected. More generally, assume that in the output of any one iteration of the column parity mixer, the A least significant bits of each output word are affected, for some $A \geq a$. Here we will denote these words as w_i, as the input to the word rotation operation (step 2). For these rotations, we choose to have the word w_0 left unchanged, while w_i for $1 \leq i < b$ are rotated as follows: every word w_i is rotated by $i \cdot A/2 + r_i$ positions, where $r_i \in [0, A/2 - 1]$. The output of step 2 is denoted as $w'_0, \ldots, w'_{b-1}$. See Figure 4 for an illustration of how each word is rotated.

These rotation amounts ensure three properties. First, the affected parts of w_{i-1} and w_i will overlap in at least one bit when added together in the next iteration, for $i = 1, \ldots, b-1$. So there cannot be any "gap" where some bit in a word will not be affected. Second, after rotations the least significant bit

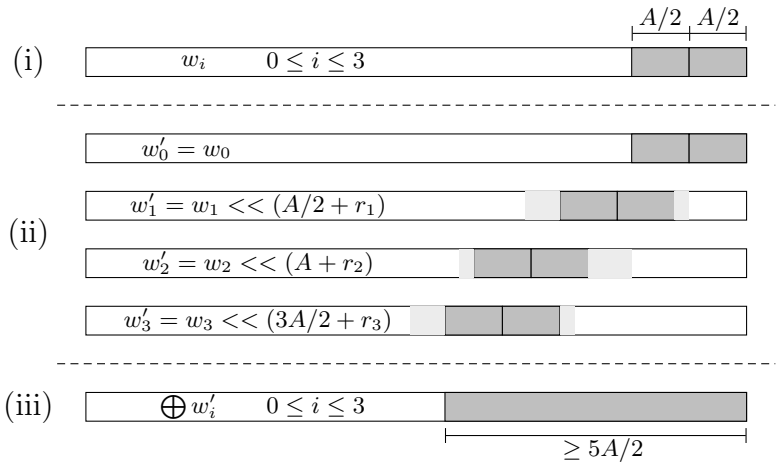


Fig. 4: Word rotation operation (step 2), applied between two iterations of the column parity mixer, acting on states with $b = 4$ words. The output words of the previous the column parity mixer, each with A least significant bits affected, are represented in region (i). The word rotation operation itself is shown in region (ii). Region (iii) represents the initial sum operation in the next iteration of the column parity mixer, with at least $5A/2$ affected bits in the input to next iteration's Θ. The block of affected bits can be anywhere in the light grey areas, depending on the values of r_i.

of the affected part of w_{i-1} cannot overlap with the affected part of w_i, for $i = 1, \ldots, b-1$. In other words, the affected parts of w_i and w_j may not overlap exactly when $i \neq j$, for any choice of r_i and r_j. Two neighboring w_i-words may therefore not cancel out when added together in the input to the next iteration. Third, the input to Θ in the next iteration will then be affected in (at least) all bits in positions $0, \ldots, (b+1)A/2$. Hence the size of the block of affected bits will increase by a factor of at least $(b+1)/2$ from one iteration to the next.

Using this strategy, the number of affected bits in $w_0, \ldots, w_{b-1}$ will grow exponentially with the number of iterations, and after $\lceil \log_{(b+1)/2}(s) \rceil$ iterations we are guaranteed the whole cipher state will be affected.

3.2 The structure in rotation-based linear layers

One can imagine many other ways of designing a linear transformation acting on a state consisting of s-bit words, using only rotations within the words and XOR additions of whole words. We show below that any linear transformation within these constraints will have a particular structure.

Assume that the state consists of $w_0, \ldots, w_{b-1}$, where each w_i is a word of s bits. We let the state block $\boldsymbol{w}$ be a binary vector of length bs, given as the concatenation of the words: $\boldsymbol{w} = (w_0 || \ldots || w_{b-1})$. Let M be the $bs \times bs$ matrix

over $\mathbb{F}_2$ that realises a rotation-based linear transformation L, such that the output of L is given as $L(\boldsymbol{w}) = \boldsymbol{w}M$.

Proposition 1. *The matrix M can be decomposed into b^2 sub-matrices $M_{i,j}$ for $0 \leq i, j \leq b-1$ of size $s \times s$ each. Let $M_{i,j}[r]$ be row r in $M_{i,j}$, for $0 \leq r \leq s-1$. Then $M_{i,j}[r] = M_{i,j}[0] << r$.*

Proof. Let the state $\boldsymbol{e}_i$ be given as the state where bit number i in $\boldsymbol{e}_i$ is 1, and all others are 0, for $0 \leq i \leq bs-1$. Then the top row of M is given as $L(\boldsymbol{e}_0)$. Whatever bits are set in $L(\boldsymbol{e}_0)$, they are all a result of the single 1-bit in $\boldsymbol{e}_0$ being added multiple times onto the words, with rotations of the words in between.

The second row of M is given as $L(\boldsymbol{e}_1)$. The exact same additions and rotations that produced $L(\boldsymbol{e}_0)$ from the single set input bit will also produce $L(\boldsymbol{e}_1)$, except everything happens shifted by one position to the left, modulo s. Hence every word in $L(\boldsymbol{e}_1)$ will be equal to the same word in $L(\boldsymbol{e}_0)$, but shifted by one position. This repeats for every row of M, so $M_{0,j}[r] = M_{0,j}[r-1] << 1$.

Row s of M is produced as $L(\boldsymbol{e}_s)$. The single set bit in the input then jumps from appearing in w_0 of the state to w_1. The word w_1 is rotated independently of w_0, so the cancellations and additions from the single set bit in $\boldsymbol{e}_s$ that occurs when producing $L(\boldsymbol{e}_s)$ are different from those that produced $L(\boldsymbol{e}_{s-1})$. Hence row s of M, and the top row of each $M_{1,j}$, will be unrelated to row $s-1$ of M. However, each row $M_{1,j}[r]$ will be rotations of $M_{1,j}[0]$ by the same reason given above. This argument repeats every time the single set bit in $\boldsymbol{e}_i$ jumps from one word to the next, and the result follows.

Proposition 1 essentially states that M can be decomposed into b^2 circulant matrices. This can also be observed by noticing that M may be considered as the binary representation of a linear transformation over the module $\mathcal{R}^b$, where $\mathcal{R}$ is the ring $\mathbb{F}_2[X]/(X^s+1)$. We provide more details in Appendix A. Also in the Appendix, Figure 9 gives an example of a matrix realising a rotation-based linear layer with $b = 5$ and $s = 329$ (as used in Fasta). For comparison, it also shows a matrix realising five parallel applications of Rasta (with same parameters). The block structure is clearly visible in the rotation-based linear transformation, whilst the comparable matrix for Rasta is a block diagonal matrix with random blocks.

4 Specification of FASTA

In this section we define Fasta, a stream cipher whose circuit for generating the keystream has been designed to be efficiently evaluated homomorphically. As the name suggests, Fasta is based on *Rasta* and is *fast* to execute when implemented in HElib using the BGV levelled homomorphic encryption scheme (see Section 2.3). We define a single instantiation of Fasta, with parameters selected to give 128 bits of security, both as a stand-alone symmetric cipher and when used in tandem with a specific instantiation of the BGV scheme it is designed

for. It is of course possible to make variants of Fasta with higher or lower security claims, but one should then also find instances of FHE schemes with the same security level and with a number of slots that matches the given variant of Fasta. Finding matching combinations of FHE parameters and symmetric designs is a study in itself, so we limit ourselves to focus on only one particular variant here. We follow Rasta's approach for setting the data limit, that at most $2^{64}/1645$ calls to Fasta with the same key can be made.[8]

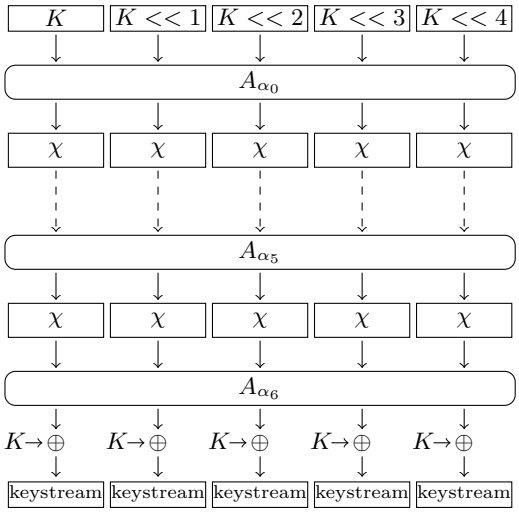


Fig. 5: High-level description of Fasta.

4.1 High-level overview

Fasta takes a 329-bit secret key K and produces 1645 bits of keystream at each call. The cipher state consists of five words $w_0, \ldots, w_4$ of 329 bits each that are initialised as $w_i = (K <<< i)$, for the secret key K. The choice of word length (329 bits) follows a search for values of m as an instatiation of BGV, that provided 128 bits of security (as FHE scheme) and gave a large, odd number of slots s. The value selected was the prime $m = 30269$, so that $\phi(m) = 30268 = 329 \times 92$, giving $s = 329$ slots (see Section 2.3). The number of state words (five) provided a good trade-off between the size of the state and the number of iterations required to generate invertible rotation-based linear layers which are expected to offer balanced diffusion.

[8] Since Fasta has a 1645-bit state, this sets the maximum length of the keystream generated under the same key to 2^{64} bits.

Each application of Fasta takes in $7 \cdot (63 + 1645) = 11956$ pseudo-random bits for specifying the particular permutation that produces a keystream block. These bits are labelled $\alpha = (\alpha_0, \ldots, \alpha_6)$, where each α_j is a 1708-bit value. In the same way as Rasta, the contents of α are pseudorandomly generated based on a counter and a nonce N which are fed into a XOF (see Figure 2).

The keystream generation applies a round function 6 times. The round function consists of an affine layer A_{α_j}, indexed by α_j for $0 \leq j \leq 5$, followed by a non-linear transformation of the cipher state. The keystream generation ends with a final affine layer A_{α_6} and a feed-forward of the secret key XORed onto each of the words. The resulting output is taken as 1645 bits of keystream. The cipher is shown in Figure 5.

4.2 The non-linear layer

The non-linear layer uses the χ-function proposed in [Dae95], which is also used in Rasta and Keccak. It is applied on each of the five words of the state in parallel as shown in Figure 5. If we label the input bits to χ as $x_0, \ldots, x_{328}$, the output bits y_i are given by

$$y_i = x_{i+1}x_{i+2} + x_i + x_{i+2},$$

where all indices are computed modulo 329.

4.3 The affine layer

Affine layers in Fasta consist of a rotation-based $\mathbb{F}_2$-linear transformation, followed by the addition of a round constant. The linear transformation is constructed as described in Section 3.1, with $b = 5$ and $s = 329$, and will consist of four iterations. A guiding principle in Rasta, which we also follow in Fasta, is that every linear transformation is pseudorandomly generated from a large family of transformations and is used only once in an instantiation of Fasta. The affine transformation we use is parameterised by a 1708-bit value α_j, which will select instances from the class of linear mappings from Section 3.1, as well as selecting the constant to be added after the linear transformation.

The Θ-function in each iteration is shown in Figure 6. It ensures that the number of affected bits in the output is increased by 9 from the number of affected bits in the input. Moreover, as b is odd, the possible choices for rotation values ensure that the resulting column parity mixer operation, and therefore the entire affine layer, is invertible.

Recall that the affected part of each word at any point is defined as the bits that may depend on the bit $x_0[0]$ at the input of the linear transformation. After the first iteration, the number of affected bits in each word will be 10. The rotations before the next iteration are therefore given as:

$$\begin{array}{l} w_1 = (w_1 << 5 + i_1),\ w_3 = (w_3 << 15 + i_3) \\ w_2 = (w_2 << 10 + i_2),\ w_4 = (w_4 << 20 + i_4) \end{array}, \text{ where } 0 \leq i_* \leq 4.$$

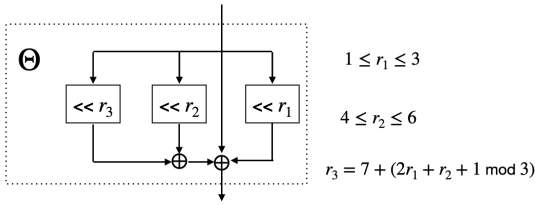


Fig. 6: The Θ-function used in FASTA's linear layer. The values of r_1 and r_2 are generated randomly from the nonce N, and the value given for r_3 ensures invertibility of the resulting column parity mixer operation.

The number of affected bits in the block going into the second Θ will therefore be at least $20 + 10 = 30$, and the number of affected bits in each word after the second iteration will be at least 39. The words $w_1, \ldots, w_4$ are then rotated by

$$\begin{array}{l} w_1 = (w_1 << 19 + j_1),\ w_3 = (w_3 << 57 + j_3) \\ w_2 = (w_2 << 38 + j_2),\ w_4 = (w_4 << 76 + j_4) \end{array}, \text{ where } 0 \leq j_* \leq 18.$$

The affected part of the word going into Θ in the third iteration will then cover at least the $39+76 = 115$ least significant bits, and the output will have at least 124 affected bits. The output is added onto every word, so the 124 least significant bits of every w_i will be affected. The words $w_1, \ldots, w_4$ are then rotated by the following amounts before going into the fourth and last iteration:

$$\begin{array}{l} w_1 = (w_1 << 62 + l_1),\ \ w_3 = (w_3 << 186 + l_3) \\ w_2 = (w_2 << 124 + l_2),\ w_4 = (w_4 << 248 + l_4) \end{array}, \text{ where } 0 \leq l_* \leq 61.$$

Note that the most significant bits of the affected part of the word w_4 (located in positions $123, 122, \ldots$) wraps around when rotated by 248 positions, as the words have length 329. This means that the entire input block to Θ in the last iteration will be affected, and after adding the output of Θ onto each w_i the entire cipher state is affected. The complete linear transformation is depicted in Figure 7.

Pseudorandomly generating the rotation-based affine layers A_{α_j}. The rotation values of the linear transformations and the constant part of the affine layers A_{α_j} are defined based on 1708-bit values α_j generated pseudorandomly (using a XOF). We take 1645 bits from α_j to define the constants value. The remaining 63 bits are used to define 24 rotation values (three used in each of the four instances of Θ and four in each of the three word rotations between Θ-iterations). Details on how this is done are given in Appendix B.

4.4 Comparing FASTA with other ciphers for hybrid encryption

Rasta is a family of stream ciphers, with the benefit of having a low and constant multiplicative depth regardless of how much keystream is produced (Section 2.2).

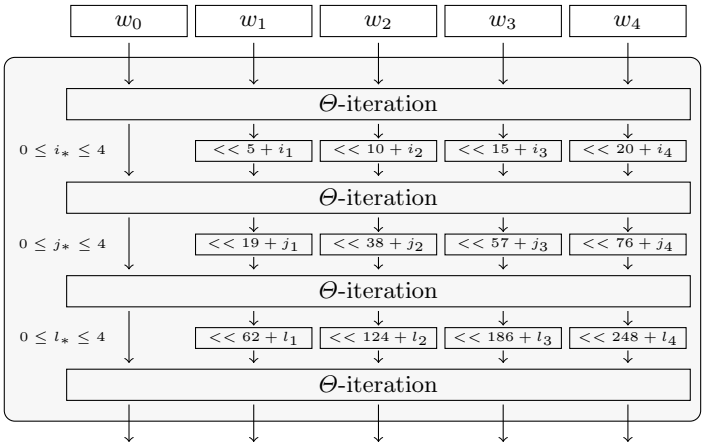


Fig. 7: The linear transformation of Fasta. The exact rotation amounts i_*, j_*, l_* and in the Θ-iterations are determined by α_j.

As discussed, Fasta is a variant of Rasta with a dedicated design for the efficient evaluation over FHE schemes. In fact, as shown in Figure 5, Fasta could be seen as five parallel calls of Rasta (with block size $s = 329$ and 6 rounds), but with one main difference. In the 5-parallel Rasta calls, the combined affine transformations A_i would be represented by a block diagonal matrix, with each 329×329 block being generated pseudorandomly. On the other hand, in Fasta the A_i are 1645×1645 rotation-based transformations, essentially tying the transformations of the five blocks together. Motivation for the choices for the value of s and the structure of A_i were given early in this section. As shown in Section 6, this structure and parameter choices will allow Fasta to be homomorphically evaluated much more efficiently in BGV/HElib, when compared to five parallel calls of Rasta.

Another drawback from Rasta is the inefficiency of requiring many random linear transformations, which need to be generated and stored. Other variants have also been proposed to address this feature. Dasta [HL20] simplifies the generation of the linear layers, by using a single fixed matrix composed with a permutation of the bits in the cipher block. These permutations are constructed by cyclically rotating smaller bit sequences that are part of the cipher block. This means much less randomness is needed from the XOF. However, rotating only part of a cipher block is difficult to achieve in a packed FHE implementation of Dasta. Thus Fasta presents the same advantages when evaluated homomorphically compared to Dasta.

Masta [HKC+20] abandons $\mathbb{F}_2^s$ as the native plaintext space, and can be seen as a Rasta variant with plaintext elements in $\mathbb{F}_p$ for a prime $p > 2$. The linear transformations of Masta are then simply chosen as a multiplication with an element chosen pseudo-randomly from $\mathbb{F}_{p^s}$, where s is the number of slots in the FHE ciphertext. The designers state that this speeds up the homomorphic

evaluation of the cipher by a factor of more than 3000 compared to Rasta. However, this comparison should be done with caution as Masta is tailored to computations on integers and Rasta (and Fasta) was designed for binary circuits.

The most recent Rasta variant proposal is called Pasta [DGH+21b]. In contrast to Fasta, Pasta has the plaintext elements taken from the field $\mathbb{F}_p$, where p is a large prime. The linear layers in Pasta are chosen in a structured way, requiring only the sampling of s random elements from $\mathbb{F}_p$ to generate an $s \times s$ matrix. All together, this leads to Pasta being up to 6 times faster than Masta in certain scenarios.

Other symmetric key ciphers proposed for hybrid encryption include LowMC [ARS+15], FLIP [MJSC16] and Kreyvium [CCF+16]. LowMC is a family of block ciphers, with multiplicative depth at least 12, while Kreyvium is a stream cipher based on Trivium. Kreyvium has the drawback that the multiplicative depth for producing keystream increases with the output length. For comparison with Fasta, producing 1645 bits of keystream with Kreyvium requires multiplicative depth of at least 17 (compared to 6 for Fasta). FLIP is also a stream cipher, with the benefit that the multiplicative depth for producing the keystream is held constant at 4. However, FLIP requires a much larger number of AND operations per bit; moreover, the successful cryptanalysis of its original version [DLR16] called for the selection of more conservative parameters. Finally, we also mention Ciminion [DGGK21], a recent proposal oriented around a large field $\mathbb{F}_q$, which aims to minimise the number of field multiplications in its design. However, while the total number of field multiplications in Ciminion might arguably be small (at least 56), they all appear sequentially. This leads to a multiplicative depth of Ciminion of at least 56, making it very unsuitable for hybrid encryption in the FHE setting.

Overall, among the ecosystem of FHE-friendly stream ciphers, Fasta has been specifically designed to improve on the efficiency of both Rasta and Dasta, while keeping the original plaintext elements as bits in $\mathbb{F}_2$.

5 Security Analysis

Fasta is a Rasta variant, which introduces a new idea for a FHE-friendly linear-layer design. Like Rasta, it also uses the $A(SA)^d$ structure, with the non-linear layer S based on the χ-transformation, and affine round transformations drawn from a large family of affine mappings. Moreover, as discussed above, Fasta can be seen as five parallel calls of a particular Rasta instance, however under the operation of different enlarged linear layers – Fasta's composed of a rotation-based transformation and the 5-parallel Rasta a block diagonal matrix, in both cases pseudo-randomly generated. As a result of these design choices, we claim that we can leverage most of the analysis originally performed for Rasta in [DEG+18] to assess the security of Fasta. For example, like Rasta we also disallow related-key attacks, and thus differential-type attacks should likewise not apply to Fasta. In this section we will therefore only discuss a subset of attacks

considered in [DEG+18], indicating when required how to adapt the original discussion to FASTA's setting. We first consider the properties of the rotation-based linear transformations introduced earlier, and used in FASTA. We also discuss the feasibility of attacks based on the algebraic structure of the cipher, and of linear approximation based attacks, again leveraging the corresponding discussions from [DEG+18].

5.1 On the structure of FASTA's linear transformation

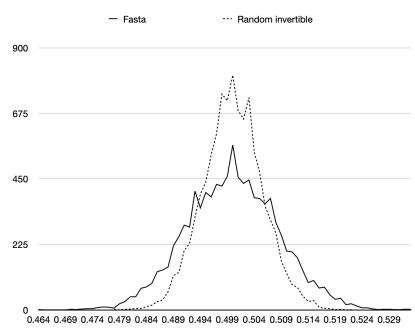
Balanced diffusion of FASTA's family of linear transformations. The Θ function used FASTA's linear layer has the property that every input bit in position i, for $0 \leq i \leq 328$, will affect the output bits in positions $i, \ldots, i+9$ mod 329. As explained in Section 4.3, the influence of $x_0[0]$ will spread to the entire cipher state after applying the linear transformation once. By rotational symmetry, this applies to every bit in the input words, so every bit in the output of the linear transformation may have any of the input bits in its linear expression.

When adding words together at the start of every iteration, some of the affected parts of the input words will overlap. As an input bit to Θ is spread to approximately half of the bits in its neighbourhood of the output, this makes approximately half of the affected part of the cipher state depend on approximately half of the input bits it depends on. In total, we therefore expect balanced diffusion for the linear layers in our family.

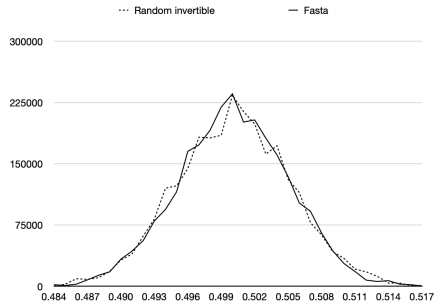
To confirm this, we have generated 10000 matrices appearing as linear transformations in FASTA, and considered their statistics compared to rotation-based random matrices. Figure 8a shows the distribution of the percentage of set bits in these matrices. The distribution is well approximated by a normal distribution with mean 50%, but have a slightly higher variance of set bits than random invertible rotation-based matrices.

In Figure 8b we have measured how well a sample of 10000 matrices satisfies balanced diffusion according to Definition 1. More precisely, for a given pair of numbers (i, j), $0 \leq i, j \leq 1644$ we measured $Pr[\langle \mathbf{e}_i L, \mathbf{e}_j \rangle = 1]$ across the 10000 matrices. We did this for all $1645 \times 1645 = 2706025$ different pairs (i, j), and counted the frequencies of probabilities seen. The plot is shown in Figure 8b, and compared against the same experiment for random invertible rotation-based matrices. As we can see, all probabilities are normally distributed around 0.5, and there is no significant difference between the matrices used in FASTA and those generated at random.

Pairwise dependence/independence of entries in the linear transformation. The FASTA state consists of 5 words with 329 bits each. Proposition 1 decomposes the linear transformation matrix M into 25 submatrices, each of size 329×329. Each of these 25 submatrices are defined in terms of their respective top row. Looking at each submatrix in isolation, each of its rows is a cyclic rotation by 1 of the row above. Let D be a submatrix of M, and $D_{i,j}$ be



(a) Distribution of the percentage of set bits in the matrices.



(b) Distribution of measured values of $Pr[\langle e_i L, e_j \rangle]$ across the matrices, for all $0 \leq i, j \leq 1644$.

Fig. 8: Statistics for 10000 matrices used in FASTA and 10000 rotation-based invertible matrices with 5×5 blocks and random top row in each block.

an entry in D. It follows from the row rotation property that $D_{i+1,j+1} = D_{i,j}$, which generalizes to $D_{0,j} = D_{a,j+a}$, for $0 \leq a, j \leq 328$ and where indices are computed modulo 329.

As M displays random behaviour and is expected to provide balanced diffusion, we will make the reasonable assumption that pairwise entries are independent, for any of the 25 submatrices in M. Furthermore, two entries from different submatrices are also treated as independent.

5.2 Algebraic attacks

Given the keystream $Z = (z_0, \ldots, z_{1644})$ produced on a call to FASTA's keystream generator for an unknown key $K = (k_0, \ldots, k_{328})$, it is possible to express the keystream bits as polynomials in $k_0, \ldots, k_{328}$ to get a set of polynomial equations over $\mathbb{F}_2$:

$$\begin{aligned} f_0(k_0, \ldots, k_{328}) + z_0 &= 0 \\ f_1(k_0, \ldots, k_{328}) + z_1 &= 0 \\ &\vdots \\ f_{1644}(k_0, \ldots, k_{328}) + z_{1644} &= 0 \end{aligned} \tag{1}$$

The attacker may repeat calls to the keystream generator to gather more such equations. The fact that new linear layers will be applied for each repetition means that new functions f_i will be used to define these fresh equations, up to 2^{64} due to the cipher's data limit. We therefore consider algebraic attacks to be the most promising cryptanalytical technique against FASTA, and consider its feasibility below.

Standard linearization-based attack. The equation system (1) forms the foundation of the standard linearization attack. In such an attack, given a system

of non-linear multivariate polynomial equations, all monomials are substituted with a new "variable", and the resulting set is considered as a system of linear equations over these variables. To fully solve this system, an attacker needs to collect as many equations as there are variables, which then allows for a unique solution to be found through Gaussian elimination. Thus, the complexity of solving such a system based on this method is directly dependent on the number of monomials in the original system.

The maximum number of different monomials we can get is dependent on the algebraic degree of each f_i. For Fasta, the algebraic degree of f_i is upper bounded by $2^6 = 64$, since the degree doubles with every application of χ and Fasta has six rounds. Thus the size of the linearized system will be at most $\sum_{i=0}^{64} \binom{329}{i} \approx 2^{535}$.

This value is computed by only considering χ in the forward direction. It is well known that the inverse of χ has high degree, but through careful study of the relationships between input and output bits to the χ operation, the authors of [LSMI21] derived further equations arising in the last round of Rasta, Dasta, and in fact Fasta. There are two important consequences of this result. Firstly, $5 \times 1645 = 8225$ equations can be derived per application of Fasta, instead of only 1645. Secondly, the last round can effectively be peeled off since the equations describing χ in the last round do not multiply inputs together, only inputs and outputs. The outputs of χ in the last round can be described as linear polynomials in $k_0, \ldots, k_{328}$, and the inputs will be polynomials of degree 32. So the number of monomials in the generated equations is reduced to at most

$$U = \sum_{i=0}^{33} \binom{329}{i} \approx 329^{33} \approx 2^{276}. \tag{2}$$

Under the assumption that all U monomials of degree up to 33 over the 329 variables are present in the system of equations, the complexity of such attack (solving a system of linear equations of size $\approx 2^{276}$) is way higher than the security level claimed for Fasta (128 bits of security). This is the behaviour we may expect for large random systems. However, for Fasta (and Rasta) we are not guaranteed that U is the number of monomials which will actually occur in the system. We explore this question in Appendix C, and conclude that the expected number of monomials appearing in the algebraic equations linking the unknowns $k_0, \ldots, k_{328}$ to the keystream bits is indeed approximated by U. Thus, similar to Rasta, linearization-based attacks are not a threat to Fasta.

Other algebraic approaches The maximum number U of monomials could be reduced by guessing g key bits, at the cost of increasing the complexity of the linearization attack by a factor of 2^g. This implies a cut-off for guessing bits at $g = 128$, where the complexity increase alone will equal the claimed security level.

Even when guessing 128 of the bits in K, we are still left with $U = \sum_{i=0}^{33} \binom{201}{i} \approx 2^{252}$ monomials to linearise. As the data complexity is limited to $2^{64}/1645 \approx 2^{54}$

calls to FASTA for a given key K, the maximum number of equations we can generate, taking [LSMI21] into account, is $5 \cdot 1645 \cdot 2^{64}/1645 \approx 2^{66.3}$. We can therefore conclude that an attacker will not be able to generate enough equations for a linearization attack to succeed.

A more advanced form of algebraic attacks is based on Gröbner basis algorithms. In this case, the cipher's non-linear system is considered in its original form, and attempted to be solved using, e.g. Faugère's F5 algorithm [Fau02]. The complexity of Gröbner basis algorithms is not fully understood for systems arising from cryptographic algorithms. Although they have been applied successfully in cryptanalysis, given the sizes involved in the FASTA system, we do not consider GB-based attacks a threat to FASTA.

5.3 Attacks based on linear approximations

To assess the feasibility of attacks based on linear approximations against FASTA, we refer to the discussion in [DEG+18, Section 3.2]. There, the authors of Rasta derive upper bounds for the correlation of linear approximations after $d = 2r$ rounds based on the properties of the χ transformation. This is done by estimating the number of active bits in the input/output of applications of χ, under the assumption the linear layers are randomly generated. For example, they conclude that Rasta with block $n = 351$ and $d = 6$ rounds is not susceptible to attacks based on linear approximations.

For FASTA, the transformations in the linear layer are not random, but rather pseudo-randomly generated among the rotation-based matrices defined in Section 3. More importantly however, the non-linear layer in FASTA consists of five parallel applications of the χ transformation. Given the diffusion properties that the linear layer is expected to feature, we expect that any linear trail over two rounds of FASTA will have a correlation of much lower magnitude than for Rasta (which would consist of six applications of χ, compared to $5 \times 6 = 30$ for FASTA). Our conclusion is therefore that, as with Rasta, attacks based on linear approximations are not feasible against the parameters chosen for FASTA.

5.4 Other classical attacks

Differential attacks, higher-order differential attacks, cube attacks, and integral attacks all try to exploit the structure of a cipher in one way or another. A differential attack looks for advantageous characteristics present in the structure, before attempting to find pairs of plaintexts which satisfy these characteristics. Higher-order attacks and cube attacks exploit the algebraic degree of the output bits of a primitive, while integral attacks make use of curated sets of plaintexts. They all have in common a need to evaluate the cryptographic primitive more than once and with different inputs. With FASTA, the circuit generating a block of keystream is only used once. Furthermore, the attacker does not have the freedom to choose different inputs to FASTA's keystream generation function, as it is always the secret key. We therefore conclude that these attacks are infeasible to execute against FASTA.

6 Homomorphic implementation of FASTA

Software libraries implementing FHE or levelled homomorphic encryption (LHE) schemes have gone through extensive development over the last years. They now appear as quite robust, well documented, and user friendly. The libraries and schemes we have considered during the design of FASTA were: HElib and PALISADE with their implementations of the BGV scheme; SEAL and PALISADE with the BFV scheme; TFHE with the torus-based FHE scheme; and PALISADE's FHEW scheme. HElib, PALISADE, and SEAL also implement the CKKS scheme, but as CKKS is an approximate LHE scheme with real numbers as the plaintext space, it is not suitable for implementing Boolean circuits.

We have designed FASTA to be fast when evaluated homomorphically, while also being based on a dedicated symmetric cipher for FHE, namely Rasta. In order to ensure fast evaluation, the parallelism offered by multiple slots in the FHE scheme is used to pack many bits of the cipher state into one FHE ciphertext. The TFHE library does not yet support such parallelism, and has therefore not been a target for the design of FASTA.

Both the BFV and BGV schemes provide ciphertexts with multiple slots, but BFV needs the number of slots to be a power of 2. Also, the BGVrns scheme will always have an even number of slots. As we use the χ-transformation in FASTA's non-linear layer, this makes BFV and BGVrns less suitable since χ is only invertible when the cipher state words going through χ have an odd number of bits. Implementing FASTA (or Rasta for that matter) in BFV using packing will then have to use *dummy slots*, i.e. slots in the FHE ciphertext that are not used, but still need to be accounted for when doing rotations, as discussed below. A symmetric cipher suitable for the BFV scheme should be designed differently, and use a set of small S-boxes in a bit-sliced fashion instead of χ as the non-linear transformation.

As the number of slots in BFV and BGVrns is much higher than 329, typically in the range 2^{13} to 2^{16} for parameters giving 128-bit security, we will only use the 329 first slots of a ciphertext $c^* = \{(c_1, c_2, \ldots, c_{329}, 0, 0, \ldots, 0)\}$, and need to do cyclic rotations over only these slots. A natural way to rotate c by a positions in the 329 first slots is to first rotate c by a positions to the right, $c_r^* = (c^* >> a)$, then by $329 - a$ positions to the left, $c_l^* = (c^* << (329 - a))$, and add the two ciphertexts:

$$c_r^* = \{(\overbrace{0, \ldots, 0, c_1, c_2, \ldots, c_{329-a}}^{329 \text{ first slots}}, c_{329-a+1}, \ldots, c_{329}, 0, \ldots, 0)\}$$

$$c_l^* = \{(\overbrace{c_{329-a+1}, \ldots, c_{329}, 0, \ldots, 0}^{329 \text{ first slots}}, 0, \ldots, 0, c_1, \ldots, c_{329-a})\}$$

$$c_l^* + c_r^* = \{(\overbrace{c_{329-a+1}, \ldots, c_{329}, c_1, \ldots, c_{329-a}}^{329 \text{ first slots}}, c_{329-a+1}, \ldots, c_{329}, 0, \ldots, 0, c_1, \ldots, c_{329-a})\}$$

This effectively does a cyclic rotation of the first 329 slots, but leaves non-zero plaintext values in the dummy slots, which need to be zeroed out to prevent

them from being shifted back in on subsequent rotations. This can be done by *masking*, multiplying with a plaintext that is 1 in the 329 first slots and zero elsewhere. Unfortunately, a plaintext-ciphertext multiplication is only somewhat cheaper in terms of noise growth than a ciphertext-ciphertext multiplication, so making a customized rotation in BFV or BGVrns to accommodate for dummy slots is simply too costly in a practical implementation.

On the other hand, the BGV scheme as implemented in HElib have instances with an odd number of slots in each ciphertext. We have therefore designed Fasta to take advantage of these features, and thus enable particularly efficient homomorphic evaluation with BGV in HElib. The basis for the BGV scheme is the cyclotomic polynomial Φ_m, where m is chosen by the user. The parameter m decides the number of slots, and together with the noise budget in fresh ciphertexts, a parameter denoted by bits in HElib, also decides the estimated security level for the instance of BGV. Searching for suitable values of m we found that $m = 30269$ gives 329 slots in HElib and a security level of just over 128 bits when bits $= 500$ (if bits is lower, the security level increases). Hence we designed Fasta to give 128-bit security in itself, and to be used with the particular instance of BGV where $m = 30269$. Running Fasta in HElib with $m = 30269$ consumes approximately 260 bits , leaving up to 240 bits more for further computations in an actual use case.

Implementing Fasta in HElib starts by encrypting the 329-bit key K five times into five different HElib ciphertexts $w_0^*, \ldots, w_4^*$ with 329 slots each. Five copies of w_i^* are then made for the feed-forward of the key at the end of Fasta. The initial rotations are done by setting $w_i^* = (w_i^* << i)$, before the first affine layer is executed using only rotations and additions of the five ciphertexts. The χ-transformation works on each w_i^* individually, and is done by making two copies of w_i^* that are rotated by 1 and 2 positions respectively: $u_1^* = (w_i^* << 1)$ and $u_2^* = (w_i^* << 2)$. The output of χ is then computed as $u_1^* \times u_2^* + w_i^* + u_2^*$, using only a single ciphertext-ciphertext multiplication. The rest of Fasta is executed homomorphically in the same way, using only rotations, additions and a single multiplication for each word in the non-linear layer of each round. Finally the initial copies of w_i^* are added to the five ciphertexts in the end to produce a block of 1645 bits of key stream encrypted under FHE.

6.1 Timings of implementations

We have made both packed and bit-sliced implementations of Fasta and Rasta in some of the libraries, and timed the execution times. The code is available at `https://github.com/Simula-UiB/Fasta`. The packed version of Rasta used mul when multiplying with random matrices in the linear layer, and the block size was modified from 351 to 329 to make the block fit exactly in the BGV ciphertext (we denote this version as Rasta* in Table 1). In addition we also ran 6-round Rasta implementations published at [DGH$^+$21a]. Parameters in the BFV and BGV schemes used were chosen to give roughly 500 bits in noise budget, for equal comparison. The timings were done on a MacBook Pro with a 2.3 GHz Intel Core i5 processor and 16 GB RAM. The results are given in Table 1.

Table 1: Amortized time (in seconds) to produce one bit of key stream when executing homomorphic implementations of Rasta and Fasta.
(Rasta* denotes the cipher with a 329-bit block.)

	Library(Scheme)	Cipher	FHE encoding	time χ	time lin. trans.	time total
Implementations from [DGH+21a]	TFHE	Rasta (6 r.)	bit-sliced	0.3640	13.902	14.266
	HElib(BGV)	Rasta (6 r.)	bit-sliced	3.079	0.510	3.589
	SEAL(BFV)	Rasta (6 r.)	bit-sliced	0.6122	0.1918	0.8040
	HElib(BGV)	Rasta (6 r.)	packed	0.0956	1.0083	1.1039
Our own implementations	PALISADE (FHEW)	Rasta (6 r.)	bit-sliced	15.73	1197.8	1213.6
	TFHE	Rasta (6 r.)	bit-sliced	0.2296	11.331	11.56
	HElib(BGV)	Rasta* (6 r.)	packed	0.0166	0.2670	0.2836
	HElib(BGV)	Fasta (6 r.)	packed	0.0166	0.0260	0.0427

Unsurprisingly, the packed implementations are faster than the bit-sliced ones encrypting only a single bit in each ciphertext. The bit-sliced implementations were all optimized with "the method of the four russians" in the matrix multiplication. In the user manual of PALISADE [PRRC21, Sec. 9.3] it is noted that both the XOR and AND gates take the same amount of time in that library's implementation of FHEW. Hence the very large number of XOR gates in the matrix multiplication of Rasta explains the extremely high execution time. Note that the packed implementation from [DGH+21a] uses the 351-bit block size specified for Rasta, and therefore needs to use masking in its operations. This explains the faster run times we have for Rasta with 329-bit block.

For the packed versions, we find that Fasta is 25 times faster than Rasta, and more than 6 times faster than the "optimized" version of Rasta where the block size fits the FHE ciphertext. The difference in runtimes for Rasta with 329-bit block and Fasta is entirely due to the linear layer of Fasta having been designed for fast execution in HElib.

7 Conclusions

The design of symmetric ciphers for use with FHE has so far focused primarily on minimising multiplicative complexity. However the libraries implementing various FHE schemes have matured over the last years, with some attractive implementation features, and are now more robust and user friendly than the early versions. This motivated us to study the implementation and homomorphic evaluation of a prominent family of FHE-friendly ciphers, Rasta, on the most well-known FHE libraries.

We found that the parameters of Rasta make it difficult to efficiently use the parallelism offered by some of the FHE schemes, namely BGV and BFV. The reason for this is that these schemes are quite inflexible when it comes to the number of slots available in a single FHE ciphertext. In the case of BFV and BGVrns, the number of slots becomes much larger than we need when these

schemes are instantiated with parameters giving 128-bit security. On the other hand, for BGV in HElib the number of slots in a single ciphertext is more in line with the block size of a symmetric cipher, but it is still determined by the m-parameter and cannot be chosen freely by the user. This led us to propose Fasta.

Our research showed that when packing the bits of the symmetric cipher state into single FHE ciphertexts, only two operations are cheap to perform: additions of full FHE ciphertexts, and cyclic rotations. Multiplications, both between two ciphertexts and between plaintext and ciphertext, are expensive and should be kept to a minimum. Moreover we also found that for efficient implementations, it is important to fit the cipher block exactly into FHE ciphertexts. Otherwise, excessive slots need to be zeroed out after rotations, which invokes multiplications with a plaintext mask.

Typical FHE-friendly symmetric designs, focusing primarily on low multiplicative complexity, appear to assume bit-sliced implementations of the cipher, where we only encrypt a single bit into each FHE ciphertext and do not need to worry about slots. They are indeed easy to implement, but these choices lead to a high run-time when evaluated homomorphically. As computational complexity is the major bottleneck for FHE it is crucial that implementations can take advantage of packing features in the main FHE libraries. Our proposal Fasta demonstrates that by taking into account the features of FHE libraries and schemes in the design process we may achieve a secure and efficient FHE-friendly symmetric cipher.

Acknowledgements. We wish to thank Joan Daemen for helpful advice and discussions on column parity mixers in the early stage of this work.

References

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 430–454, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[BDPA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak reference, version 3.0, 14 January 2011. `https://keccak.team/files/Keccak-reference-3.0.pdf`.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption Without Bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325, New York, NY, USA, 2012. ACM.

[CCF+16] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream Ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression. In Thomas Peyrin, editor, *Fast Software Encryption*, volume 9783 of *Lecture Notes in Computer Science*, pages 313–333, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[CHK20] Jung Hee Cheon, Kyoohyung Han, and Duhyeong Kim. Faster Bootstrapping of FHE over the Integers. In Jae Hong Seo, editor, *Information Security and Cryptology – ICISC 2019*, pages 242–259. Springer International Publishing, 2020.

[CIM16] Georgieva M. Chillotti I., Gama N. and Izabachène M. Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In Takagi T. Cheon J., editor, *Advances in Cryptology – ASIACRYPT 2016*, volume 10031 of *Lecture Notes in Computer Science*, pages 3–33. Springer, Berlin, Heidelberg, 2016.

[Dae95] Joan Daemen. *Cipher and hash function design, strategies based on linear and differential cryptanalysis, PhD Thesis*. K.U.Leuven, 1995. `http://jda.noekeon.org/`.

[DEG+18] Christoph Dobraunig, Maria Eichlseder, Lorenzo Grassi, Virginie Lallemand, Gregor Leander, Eik List, Florian Mendel, and Christian Rechberger. Rasta: A Cipher with Low ANDdepth and Few ANDs per Bit. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, pages 662–692. Springer International Publishing, 2018.

[DGGK21] Christoph Dobraunig, Lorenzo Grassi, Anna Guinet, and Daniël Kuijsters. Ciminion: Symmetric Encryption Based on Toffoli-Gates over Large Finite Fields. Cryptology ePrint Archive, Report 2021/267, 2021. `https://ia.cr/2021/267`.

[DGH+21a] Christoph Dobraunig, Lorenzo Grassi, Lukas Helminger, Christian Rechberger, Markus Schofnegger, and Roman Walch. Framework for hybrid homomorphic encryption. `https://github.com/IAIK/hybrid-HE-framework`, 2021.

[DGH+21b] Christoph Dobraunig, Lorenzo Grassi, Lukas Helminger, Christian Rechberger, Markus Schofnegger, and Roman Walch. Pasta: A case for hybrid homomorphic encryption. Cryptology ePrint Archive, Report 2021/731, 2021. `https://ia.cr/2021/731`.

[DLR16] Sébastien Duval, Virginie Lallemand, and Yann Rotella. Cryptanalysis of the FLIP Family of Stream Ciphers. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016*, pages 457–475, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 617–640. Springer Berlin Heidelberg, 2015.

[Fau02] Jean Charles Faugère. A new efficient algorithm for computing gröbner bases without reduction to zero (f5). In *ISSAC'02: Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, pages 75–83, July 2002.

[Gen09] Craig Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

[HKC+20] Jincheol Ha, Seongkwang Kim, Wonseok Choi, Jooyoung Lee, Dukjae Moon, Hyojin Yoon, and Jihoon Cho. Masta: An HE-Friendly Cipher Using Modular Arithmetic. *IEEE Access*, 8:194741–194751, 2020.

[HL20] Phil Hebborn and Gregor Leander. Dasta – Alternative Linear Layer for Rasta. *IACR Transactions on Symmetric Cryptology*, 2020(3):46–86, 2020.

[HPS18] Shai Halevi, Yuriy Polyakov, and Victor Shoup. An Improved RNS Variant of the BFV Homomorphic Encryption Scheme. Cryptology ePrint Archive, Report 2018/117, 2018. https://eprint.iacr.org/2018/117.

[HS18] Shai Halevi and Victor Shoup. Faster Homomorphic Linear Transformations in HElib. Cryptology ePrint Archive, Report 2018/244, 2018. https://eprint.iacr.org/2018/244.

[HS20] Shai Halevi and Victor Shoup. Design and implementation of HElib: a homomorphic encryption library. Cryptology ePrint Archive, Report 2020/1481, 2020. https://eprint.iacr.org/2020/1481.

[LSMI21] Fukang Liu, Santanu Sarkar, Willi Meier, and Takanori Isobe. Algebraic Attacks on Rasta and Dasta Using Low-Degree Equations. Cryptology ePrint Archive, Report 2021/474, 2021. https://eprint.iacr.org/2021/474.

[MJSC16] Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. Towards Stream Ciphers for Efficient FHE with Low-Noise Ciphertexts. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, volume 9665 of *Lecture Notes in Computer Science*, pages 311–343. Springer Berlin Heidelberg, 2016.

[PAL] PALISADE –An Open-Source Lattice Crypto Software Library. https://palisade-crypto.org/.

[PRRC21] Yuriy Polyakov, Kurt Rohloff, Gerard W. Ryan, and Dave Cousins. PALISADE Lattice Cryptography Library User Manual (v1.11.2), 2021. https://eprint.iacr.org/2018/117.

[RAD78] R L Rivest, L Adleman, and M L Dertouzos. On Data Banks and Privacy Homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

[RDP+96] Vincent Rijmen, Joan Daemen, Bart Preneel, Antoon Bosselaers, and Erik De Win. The cipher SHARK. In Dieter Gollmann, editor, *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, volume 1039 of *Lecture Notes in Computer Science*, pages 99–111. Springer, 1996.

[SD18] Ko Stoffelen and Joan Daemen. Column Parity Mixers. *IACR Transactions on Symmetric Cryptology*, 2018(1):126–159, Mar. 2018.

[SEA20] Microsoft SEAL (release 3.6). https://github.com/Microsoft/SEAL, November 2020. Microsoft Research, Redmond, WA.

A Matrix structure of rotation-based linear transformations

To observe and study the structure of rotation-based linear transformation matrices introduced in Section 3.1, we recall the steps for constructing a rotation-based linear transformation acting on b s-bit words $w_0, \ldots, w_{b-1}$.

1. Define a column parity mixer based on a Θ operation using rotations of low amounts (compared to the word length s; see Figure 3).
2. Apply rotations to the words w_i between applications of the column parity mixer.
3. Iterate applications of column parity mixers with rotations in between, as much as needed until the entire cipher state is affected.

To describe the structure of (binary) matrices defined as above, it is helpful to consider rotation-based linear transformations as operations over the module $\mathcal{R}^b$, where $\mathcal{R}$ is the ring $\mathbb{F}_2[X]/(X^s+1)$. In this case, each w_i can be considered as a polynomial $w_i(X) = a_{s-1}X^{s-1} + \ldots + a_2X^2 + a_1X + 1$, where $a_j \in \mathbb{F}_2$. Note that the XOR operation of two words w_i, w_j corresponds to addition in $\mathcal{R}$, while the rotation operation $w_i << r$ corresponds to the multiplication of $w_i(X)$ by X^r.

Then let $\boldsymbol{w} = (w_0, \ldots, w_{b-1}) \in \mathcal{R}^b$ be the input of a rotation-based linear transformation L defined as above. The application of a column parity mixer based on a Θ operation using rotations/XORs (step 1) corresponds to:

(i) $(w_0, \ldots, w_{b-1}) \mapsto (w_0 + \ldots + w_{b-1}) = w \in \mathcal{R}$
(ii) $w \mapsto w \cdot p_\Theta$, where $p_\Theta \in \mathcal{R}$ is a polynomial defined by the rotations and XOR operations in Θ.
(iii) $w \cdot p_\Theta \mapsto (w_0 + w \cdot p_\Theta, \ldots, w_{b-1} + w \cdot p_\Theta) \in \mathcal{R}^b$.

Thus application of a column parity mixer operation on $\boldsymbol{w} = (w_0, \ldots, w_{b-1}) \in \mathcal{R}^b$ can be represented as a matrix over $\mathcal{R}$ given by

$$P_\Theta = \begin{pmatrix} p_\Theta + 1 & p_\Theta & \ldots & p_\Theta \\ p_\Theta & p_\Theta + 1 & \ldots & p_\Theta \\ \ldots & \ldots & \ldots & \ldots \\ p_\Theta & p_\Theta & \ldots & p_\Theta + 1 \end{pmatrix}$$

Likewise, the application of rotations $<< r_i$ to the individual words w_i of the state (step 2) can be represented as a matrix

$$R_v = \begin{pmatrix} X^{r_0} & 0 & \ldots & 0 \\ 0 & X^{r_1} & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & X^{r_{b-1}} \end{pmatrix},$$

where $v = (r_0, r_1, \ldots, r_{b-1})$. These two operations are then iterated n times, using different Θ_i and word rotations $v_i = (r_0, \ldots r_{b-1})$ (step 3). It follows that

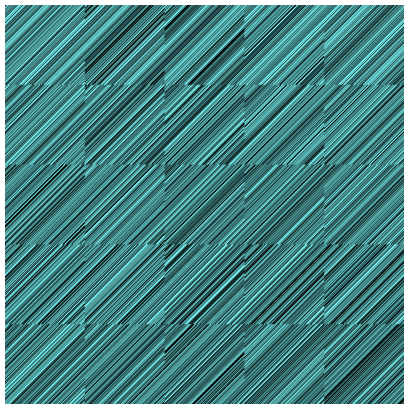
the matrix M representing a rotation-based linear transformation over $\mathcal{R}^b$ can be defined as

$$M = P_{\Theta_1} \cdot R_{v_1} \cdot P_{\Theta_2} \cdot R_{v_2} \cdot \ldots \cdot R_{v_{n-1}} \cdot P_{\Theta_n}$$

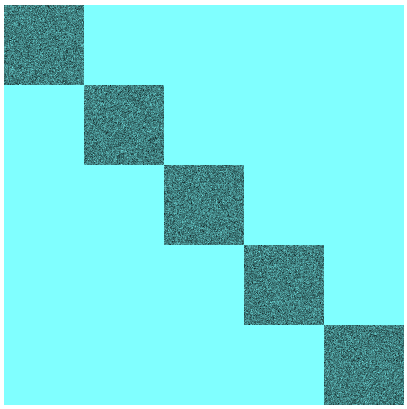
Every entry of M is a univariate polynomial of degree at most $s-1$. Note that the multiplication of $w_i \in \mathcal{R}$ by a polynomial $p \in \mathcal{R}$, when considered as a $\mathbb{F}_2$-linear transformation, can be represented as a binary circulant matrix. It follows that, when considered as a $\mathbb{F}_2$-linear transformation acting on the state block $\boldsymbol{w} \in (\mathbb{F}_2)^{bs}$, the $bs \times bs$ matrix M realising a rotation-based linear transformation L, with $L(\boldsymbol{w}) = \boldsymbol{w}M$, can be decomposed into b^2 sub-matrices as described in Proposition 1.

For example in Fasta, we have $b = 5$ and $s = 329$. Moreover, Θ can be realised by multiplication by the polynomial $p_\Theta = X^{r_3} + X^{r_2} + X^{r_1} + 1$ (where $1 \leq r_1 \leq 3$, $4 \leq r_2 \leq 6$, and $7 \leq r_3 \leq 9$; refer to Figure 6), and the word rotation operations R_v are defined as given in Figure 7. Four iterations are required to generate the matrix M. As discussed in Section 4, these choices ensure that the matrices P_Θ, R_v, and as consequence M, are invertible. An example of such a matrix M generated following this method can be seen in Figure 9a. Each of the 25 blocks is a 329×329 circulant matrix over $\mathbb{F}_2$.

For the purpose of comparison, we also include the matrix for a linear transformation realising five parallel calls to Rasta with same parameters (Figure 9b). In this case, the resulting linear transformation can be represented as a block diagonal matrix, with random 329×329 sub-matrices in the diagonal, and all zero matrices elsewhere.



(a) Matrix realising a rotation-based linear transformation with 5 words of length 329.



(b) Matrix for linear layer tying five parallel applications of Rasta together.

Fig. 9: Structure of matrices for Fasta and five parallel calls to Rasta. Black pixels indicate 1-bits and blue pixels are 0-bits.

B Mapping α_j to rotation values and round constants

Let $r_1^{(t)}$, $r_2^{(t)}$ and $r_3^{(t)}$ be the rotation amounts used in Θ in iteration t, for $1 \leq t \leq 4$. There are then 24 rotation amounts that need to be decided from α_j. The $r_1^{(t)}$ and $r_2^{(t)}$ can take 3 values each, and $r_3^{(t)}$ is computed from these, for a total of nine different instances of Θ. Each of the four i_*, j_*, l_* can take 5, 19, and 62 values each, respectively. There are therefore $T = 3^8 \cdot 5^4 \cdot 19^4 \cdot 62^4 \approx 2^{62.78}$ different instances in the class $\mathcal{L}$ of rotation-based linear transformations we have defined.

We split α_j into $\alpha_j = (\alpha_j^r, \alpha_j^c)$, where α_j^r is 63 bits and α_j^c is 1645 bits. The 24 rotation values are computed from α_j^r, as in Algorithm 1. Apart from the $r_3^{(t)}$ values, what we are essentially doing is first computing $B = \alpha_j^r \mod T$, and then writing B in a mixed base: the eight least significant digits in base 3, the next four digits in base 5, the next four in base 19, and the four most significant digits in base 62. Keeping in mind that $r_1^{(t)}$ and $r_2^{(t)}$ will have 1 and 4 added to them, the rotation amounts can then be read out as the digits of B, written in this mixed base:

$$\begin{aligned} B = {} & k_3 \cdot 62^3 \cdot 19^4 \cdot 5^4 \cdot 3^8 + k_2 \cdot 62^2 \cdot 19^4 \cdot 5^4 \cdot 3^8 + \ldots \\ & + r_2^{(2)} \cdot 3^5 + r_2^{(1)} \cdot 3^4 + r_1^{(4)} \cdot 3^3 + r_1^{(3)} \cdot 3^2 + r_1^{(2)} \cdot 3 + r_1^{(1)}. \end{aligned}$$

After applying the linear transformation, the 1645-bit value α_j^c is XORed onto the state to produce the affine layer output.

Algorithm 1: Determining rotation amounts from α_j^r.

Result: Rotation amounts for linear transformation are fixed.

$B \leftarrow \alpha_j^r \mod T$
for $t = 1$ to 4 **do**
  $r_1^{(t)} \leftarrow 1 + (B \bmod 3)$
  $B \leftarrow \lfloor B/3 \rfloor$
end for
for $t = 1$ to 4 **do**
  $r_2^{(t)} \leftarrow 4 + (B \bmod 3)$
  $r_3^{(t)} \leftarrow 7 + (2r_1^{(t)} + r_2^{(t)} + 1 \bmod 3)$
  $B \leftarrow \lfloor B/3 \rfloor$
end for
for $t = 1$ to 4 **do**
  $i_t \leftarrow B \bmod 5$
  $B \leftarrow \lfloor B/5 \rfloor$
end for
for $t = 1$ to 4 **do**
  $j_t \leftarrow B \bmod 19$
  $B \leftarrow \lfloor B/19 \rfloor$
end for
for $t = 1$ to 4 **do**
  $l_t \leftarrow B \bmod 62$
  $B \leftarrow \lfloor B/62 \rfloor$
end for

C Standard linearization-based attack against FASTA

We examine the question of the number of monomials actually occurring in an algebraic description of FASTA, following a similar discussion from [DEG+18].

Let M be the matrix over $\mathbb{F}_2$ that realises one of FASTA's rotation-based linear transformations, let $x = (x_0, \ldots, x_{1644})$ be the input state and $A(x) = M \cdot x + c$. From the description of χ in the non-linear layer S, one round $S \circ A(x)$ of FASTA can be described by the following equations (from [DEG+18]):

$$S \circ A(x)_i = \sum_{j=0}^{k-1} \sum_{l=j+1}^{k-1} a_{j,l}^i \cdot x_j \cdot x_l + \sum_{j=0}^{k-1} b_j^i \cdot x_j + g^i, \tag{3}$$

where i denotes the polynomial representing the i-th bit in the cipher block after $S \circ A(x)$. As the word size is 329, $i+1$ and $i+2$ "wrap around", i.e. they are calculated as $i-328$ and $i-327$ when $i \bmod 329 = 328$ and 327. The coefficients of $S \circ A(x)_i$ are given by

$$\begin{aligned} a_{j,l}^i &= M_{i+1,j} \cdot M_{i+2,l} + M_{i+2,j} \cdot M_{i+1,l}, \\ b_j^i &= M_{i,j} + c_{i+2} \cdot M_{i+1,j} + (1 + c_{i+1}) \cdot M_{i+2,j}, \\ g^i &= c_i + c_{i+2} + c_{i+1} \cdot c_{i+2}. \end{aligned}$$

We can see that the term containing the coefficient $a_{j,l}^i$ contains the only multiplication, meaning it is the only place where the algebraic degree may increase. We only need $a_{j,l}^i = 1$ for at least one i for the corresponding monomial to be present in the output. We first find the probability that each coefficient $a_{j,l}^i$ is 0. From the above equations we get

$$P[a_{j,l}^i = 0] = P[M_{i+1,j}M_{i+2,l} = M_{i+2,j}M_{i+1,l} = 0] + P[M_{i+1,j}M_{i+2,l} = M_{i+2,j}M_{i+1,l} = 1] \tag{4}$$

In Section 5.1, we found when two entries in M are equal with certainty, due to the rotational structure in M, and when they are considered independent. Put into context of Equation 4, we have that two entries $M_{i+1,j}$ and $M_{i+2,l}$ are equal when

$$l = \begin{cases} j+1 & \text{for } j \neq 328 \mod 329 \\ j-328 & \text{for } j = 328 \mod 329 \end{cases}$$

Otherwise, $M_{i+1,j}$ and $M_{i+2,l}$ are considered as independent in our analysis.

The equal entries are split into two cases, depending on whether j or l are crossing from one sub matrix to another or not, i.e., to handle "wrap-around" of sub-matrices.

We expect each entry in M to be present with probability one half, following the discussion in Section 5.1. This allows us to calculate $P[a_{j,l}^i = 0]$. We begin with the case where the two entries from M are equal, i.e, in general when

$l = j + 1$:

$$\begin{aligned} P[a^i_{j,j+1} = 0] &= P[M_{i+1,j}M_{i+2,j+1} = M_{i+2,j}M_{i+1,j+1} = 0] \\ &+ P[M_{i+1,j}M_{i+2,j+1} = M_{i+2,j}M_{i+1,j+1} = 1] \\ &= \frac{1}{2} \cdot \frac{3}{4} + \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{2}. \end{aligned}$$

For all independent entries, we get instead:

$$P[a^i_{j,l} = 0] = \left(\frac{3}{4}\right)^2 + \left(\frac{1}{4}\right)^2 = \frac{5}{8}.$$

This last result is the same as expected for any two entries in a random matrix. It follows that the probability that all the coefficients for the product $x_j \cdot x_l$ are equal to 0 can be estimated as

$$P[a^i_{j,l} = 0,\ \forall i = 0, \ldots, 328] \leq \left(\frac{5}{8}\right)^{329}.$$

In other words, at least one of these coefficients are 1 with probability at least $1 - \left(\frac{5}{8}\right)^{329}$.

If we consider the monomials of degree 2, it follows that we can expect an average number of monomials in each word w_i of degree 2 to be at least

$$\binom{329}{2} \cdot \left(1 - \left(\frac{5}{8}\right)^{329}\right) \simeq \binom{329}{2}.$$

We can use the same reasoning we used for monomials of degree 1, resulting in an expected number of these monomials to be $329 \cdot (1 - 2^{-329}) \approx 329$. This argument can also be applied for monomials of higher degrees. We therefore conclude that the expected number of monomials appearing in the algebraic equations linking the unknowns $k_0, \ldots, k_{328}$ to the keystream bits is approximated by U, the maximum possible number of monomials.