Chaghri — an FHE-friendly Block Cipher

Tomer Ashur[1,2] Mohammad Mahzoun[2]
tomer.ashur@esat.kuleuven.be m.mahzoun@tue.nl
Dilara Toprakhisar [1]
Dilara.Toprakhisar@esat.kuleuven.be

[1]imec-COSIC, KU Leuven, Belgium
[2] Eindhoven University of Technology, Netherlands

Abstract

The Recent progress in practical applications of secure computation protocols has also attracted attention to the symmetric-key primitives underlying them. Whereas traditional ciphers have evolved to be efficient with respect to certain performance metrics, advanced cryptographic protocols call for a different focus. The so called arithmetic complexity is viewed through the number and layout of non-linear operations in the circuit implemented by the protocol. Symmetric-key algorithms that are optimized with respect to this metric are said to be algebraic ciphers. Previous work targeting ZK and MPC protocols delivered great improvement in the performance of these applications both in lab and in practical use. Interestingly, despite its apparent benefits to privacy-aware cloud computing, algebraic ciphers targeting FHE did not attract similar attention.

In this paper we present CHAGHRI, an FHE-friendly block cipher enabling efficient transciphering in BGV-like schemes. A complete CHAGHRI circuit can be implemented using only 16 multiplications, 48 Frobenius automorphisms and 32 rotations, all arranged in a depth-32 circuit. Our HElib implemention achieves a throughput of 0.28 seconds-per-bit which is 63% faster than *AES* in the same setting.

1 Introduction

Traditional block ciphers are built with carefully chosen linear and non-linear layers to resist well studied attacks. Besides being secure, traditional block ciphers are designed to be efficient in their hardware and software implementations. Depending on the target application domain, their design optimizes the running time, gate count, or memory/power consumption. For instance, while an IoT device calls for lower memory/power consumption and gate count, a high speed router calls for lower latency. Different efficiency metrics come into consideration when the target application domain is a secure computation protocol. *Multi-Party Computation* (MPC), *Zero-Knowledge* (ZK) *proofs*, and *Fully Homomorphic Encryption* (FHE) are

examples of such secure computation protocols that are described via algebraic operations. These operations can be translated into arithmetic computations and vice versa. Converting computations into a sequence of algebraic operations over a finite field is called *arithmetization* and it was first applied to cryptographic protocols by Lund *et al.* [29].

Consider the following scenario in which a secure computation protocol employs a block cipher: a client sends its data encrypted under an FHE scheme to a cloud server that operates on encrypted data. However, depending on the complexity of the function performed by the server, the scheme's parameter set might result in a drastic increase in the size of the freshly encrypted ciphertext. Consequently, this increase would add unwanted overhead to the communication. One solution to this problem is *transciphering* meaning all the private data sent by a client can be encrypted using a block cipher. Then, the server decrypts homomorphically, and consequentially they are able to operate on encrypted data without additional overhead to the communication [32].

The increase in the popularity of advanced cryptographic protocols gave rise to new designs known as algebraic ciphers such as *MiMC* [2],*LowMC* [1], *Kreyvium* [7], *FLIP* [31], *Rasta*[12], *Dasta* [21], *Pasta* [13], *Fasta* [8], *Elisabeth* [9], *Rubato* [20], *Poseidon* [18], *Vision*, and *Rescue* [3]. Unlike traditional block ciphers, the design of these algorithms is driven by arithmetic complexity improving the efficiency of the protocol employing them. Therefore, the relevant attacks and security of these algorithms are also different.

As the design of algebraic ciphers is an evolving research area, there are several design strategies introduced as a framework. The *Marvellous* design strategy [3] and the *Hades* design strategy [19] are examples of such design strategies. The ciphers that are proposed following these design strategies are shown to be efficient in ZK and MPC applications. Although numerous algebraic ciphers were proposed for ZK and MPC applications, there are not many algebraic ciphers proposed in the context of FHE. Yet, FHE is an effective tool to remove privacy barriers obstructing data sharing. Therefore, designing an FHE-friendly algebraic cipher still stands as a research area that needs to be improved.

In this work we address the key factors affecting the efficiency and present a novel FHE-friendly algebraic cipher CHAGHRI. Implemented using the HElib software library CHAGHRI performs 63% faster than *AES*, making it, to the best of our knowledge, the most efficient block cipher in this setting.

This paper is structured as follows: in Section 2 we recall the *Marvellous* design strategy, the ciphers designed following it, the notion of non-procedural computation, and BGV-based fully homomorphic encrytpion. In Section 3 we motivate the decisions taken in designing CHAGHRI, and in Section 4 we give the specifications of the cipher. Following this, we argue the security and efficiency of CHAGHRI by presenting a security analysis against the applicable statistical and structural attacks in Section 5, and the performance numbers together with the benchmarking figures in Section 6. This paper does not include a "conclusion" section.

2 Preliminaries

In this section we mention some related previous work, and some prior knowledge that is required for the design of CHAGHRI.

2.1 The Marvellous Design Strategy

The *Marvellous* design strategy [3] introduces a set of decisions to be taken when designing a secure and efficient algebraic cipher. The state of a *Marvellous* design is an element in the vector space $\mathbb{F}_q^\ell$, with q either a power of 2 or a prime number and $\ell > 1$. A *Marvellous* design is an SP network that repeatedly applies its round function to its state for N iterations. Figure 1 depicts a schematic description of the encryption operation of a *Marvellous* design. A plaintext and a master key are the inputs to the first round. Each round consists of two steps and each step employs three layers: S-box, linear, and subkey injection. The subkeys used in subkey injection are derived from the master key by means of a key schedule algorithm.

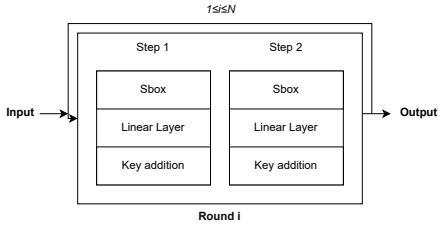

Figure 1: The encryption operation of *Vision* and *Rescue*

The S-box layer of a *Marvellous* round applies an S-box to each of the ℓ state elements. Each S-box consists of a power map $g : x^\alpha$ and possibly followed by an invertible affine transformation. The motivation behind employing a power map S-boxes is their well studied cryptanalytic properties [33]. The two steps of a *Marvellous* round employ different S-boxes in terms of their degrees. Let the S-box employed in the first step be denoted by θ_0, and the S-box employed in the second step be denoted by θ_1. θ_0 is chosen such that it has a high degree when the encryption is performed and a low degree when the decryption is performed. θ_1 is chosen such that it serves the opposite goal: it has a low degree when the encryption is performed and a high degree when the decryption is performed. This construction provides a high degree in both encryption and decryption, and consequently results in the same cost for both.

The linear layer diffuses local properties to the entire state. This is realized by multiplying the *Marvellous* state vector by a maximum distance separable (MDS) matrix. The authors [3] offer to use $\ell \times 2\ell$ Vandermonde matrices using powers of an $\mathbb{F}_q$ primitive element. To obtain the MDS matrix, the Vandermonde matrix is echelon reduced and the $\ell \times \ell$ identity matrix is removed.

The key schedule algorithm of a *Marvellous* design is indeed the iteratively applied encryption round function. In order to generate the subkeys, the round function takes

the master key instead of the plaintext input, and takes additional round constants instead of the subkeys injected. The round constants are chosen such that they do not belong to a subfield of $\mathbb{F}_q$, nor are rotational invariant. The intermediate state after the round constant injection is provided as a subkey.

The number of rounds in a *Marvellous* round is set to be

$$2 \cdot \max(r_0, r_1, 5),$$

where r_0 is set to be the maximum number of rounds that can be attacked by differential and linear cryptanalysis, higher-order differentials and interpolation attacks; r_1 is said to be the instance-specific number of rounds that can be attacked by a Gröbner basis attack. Five is the sanity factor that protects the cipher against redundant optimization attempts weakening it. As a result, any *Marvellous* instance is set with a minimum of 10 rounds.

Vision. *Vision* is a *Marvellous* family operating on binary fields with its native field $\mathbb{F}_{2^n}$. Most aspects of *Vision* are directly derived from the *Marvellous* design strategy. The *Vision*-specific design decisions are limited to the S-box layer which consists of an inversion (with 0 mapped to 0) followed by an affine transformation. It is constructed by first choosing a 4^{th} degree $\mathbb{F}_2$-linearized affine polynomial $B(x)$. Then,

$$\theta_1 : \mathbb{F}_{2^n} \mapsto \mathbb{F}_{2^n} : x \mapsto B(x^{-1}),$$

and

$$\theta_0 : \mathbb{F}_{2^n} \mapsto \mathbb{F}_{2^n} : x \mapsto B^{-1}(x^{-1}).$$

Rescue. *Rescue* is another *Marvellous* family, this time operating on $\mathbb{F}_p$ where p is an odd prime instead of a power of 2. Same as *Vision,* most aspects of *Rescue* are directly derived from the *Marvellous* design strategy. The S-box layer of *Rescue* consists of a power map only. It is constructed by first finding the smallest prime α such that $\gcd(p-1, \alpha) = 1$. Then,

$$\theta_0 : \mathbb{F}_p \mapsto \mathbb{F}_p : x \mapsto x^{1/\alpha},$$

and

$$\theta_1 : \mathbb{F}_p \mapsto \mathbb{F}_p : x \mapsto x^{\alpha}.$$

Rescue-Prime. *Rescue-Prime* [35] is an algebraic hash function inspired by *Rescue*. In *Rescue-Prime* the derivation of round constants is changed, the security margin is reduced from 100% to 50%, and the order of S-Boxes is swapped. Since we propose an algebraic cipher operating on a binary field, algorithm specific properties of *Rescue-Prime* are omitted in this paper. Interested readers are referred to [35] for the complete description of the algorithm.

2.2 Fully Homomorphic Encryption (FHE)

Fully homomorphic encryption (FHE) is an advanced cryptographic protocol that allows users to evaluate any circuit on encrypted data without first decrypting it. FHE is an effective solution to securely outsourcing computations. However, depending on the size of the computation, the data might be drastically expanded when encrypted under an FHE algorithm. In this case, recalling the example given in Section 1, *transciphering* combining FHE and symmetric encryption allows an efficient encrypted data communication and computation outsourcing.

2.2.1 Brakerski-Gentry-Vaikuntanathan (BGV) Scheme

BGV is a leveled FHE scheme proposed by Brakerski, Gentry and Vaikuntanathan [5]. Leveled FHE is more restricted than FHE in that the depth of circuits it can evaluate is bounded by the parameters of the scheme. BGV uses *modulus-switching* introduced by Brakerski and Vaikuntanathan [6] to keep the noise under a threshold. Modulus switching is proposed in [6] to be applied once to obtain a ciphertext with less noise. However, it is iteratively applied in BGV to keep the noise under a certain threshold.

In this work we use a BGV variant proposed by Gentry, Halevi and Smart [16] where both ciphertexts and secret keys are represented as vectors over a polynomial ring $\mathbb{A}$, and the plaintext space is all polynomials over $\mathbb{A}_p$ for $p \geq 2$ defined by cyclotomic polynomials $\Phi_m(X)$. Additionally, at any point during the homomorphic evaluation, there are *current integer modulus q* and *current secret key s* that evolve as the homomorphic operations are applied. Decryption is done by taking the inner product of the ciphertext c and the current secret key s over $\mathbb{A}_q$. Then the result is reduced modulo p:

$$a \leftarrow [\, [\langle c, s\rangle \mod \Phi_m(X)]_q \,]_p. \tag{1}$$

($[\langle c, s\rangle \mod \Phi_m(X)]_q$: noise)

`Addition`, `multiplication` and `automorphism` are used to evaluate circuits and therefore, alter the data encrypted under these ciphertexts. `Key-switching` and `modulus-switching` are used to control the complexity of the evaluation and therefore, do not affect the underlying data.

Addition. Homomorphic `addition` is simply performed by means of a vector addition over $\mathbb{A}_q$ (with respect to the same secret key and modulus q). This operation slightly increases the noise of the ciphertext, and does not change the current secret key and the current modulus.

Multiplication. Homomorphic `multiplication` is performed by means of a tensor product over $\mathbb{A}_q$. If the two arguments of this operation have dimension n over $\mathbb{A}_q$, the output then has dimension n^2. The change in the dimension of the ciphertext consequently results in a change in the dimension of the secret key. This is because the output ciphertext would then be valid with respect to the secret key s' of dimension n^2. Therefore, the operation changes the current secret key, but not the current modulus. Homomorphic `multiplication` significantly increases the noise of the ciphertext.

Automorphism. `Automorphism` maps a polynomial $a(X) \in \mathbb{A}$ to $a^{(i)}(X) = a(X^i) \bmod \Phi_m(X)$. The set of transformations $\{a \mapsto a^i : i \in (\mathbb{Z}/m\mathbb{Z})^*\}$ forms a group under the composition operation, and this group is isomorphic to $(\mathbb{Z}/m\mathbb{Z})^*$. Let c be a valid ciphertext encrypting a with respect to s and q. Then the output of the `automorphism` operation $c^{(i)}$ is a valid ciphertext encrypting $a^{(i)}$ with respect to $s^{(i)}$ and q. Different than the `addition` and the `multiplication`, this operation does not increase the noise of the ciphertext.[1]

Key-switching and modulus-switching. `Key-switching` is used after the operations increasing the dimension of the secret key and `Modulus-switching` is applied to reduce the noise of the ciphertext.

Packed ciphertexts. This FHE scheme allows performing operations on packed ciphertexts. Smart and Vercauteren [34] proposed using the Chinese Remainder Theorem to represent the plaintext space $\mathbb{A}_p$ as a vector of plaintext slots. This applies when $\Phi_m(X)$ factors modulo p into l irreducible polynomials such that $\Phi_m(X) = \prod_{j=1}^{l} F_j(X) \bmod p$. Then, a plaintext polynomial $a(X) \in \mathbb{A}_p$ can be represented as encoding l different plaintext polynomials with $a_j = a \bmod F_j$. `Addition` and `multiplication` operations are then performed slot-wise. However, this is not the case for `automorphism`. If i is a power of two, then the transformation $a \mapsto a^{(i)}$ can be realized for each slot separately, and this transformation is called a `Frobenius automorphism`. Conversely, if i is not a power of two, then the transformation acts as a shift operation between the different slot elements.

2.3 Non-procedural Computation

Procedures simply consist of a series of computational steps. In procedural computation, the system's state at any point in time is a function of the system's state at a previous point in time. However, the algebraic operations employed by the advanced cryptographic protocols are better interpreted with respect to an alternative timeline. This interpretation is said to be a non-procedural computation. For instance, masked operations in MPC offer non-procedural properties by referring certain computations to an offline phase.

Non-procedural computations allow constant-time execution in operations that would otherwise have resulted in variable running time when the size of the native field changes. Therefore, exploiting non-procedural computation can improve the efficiency of advanced cryptographic protocols. In addition to efficiency, employing non-procedural computations can offer security properties without the need to increase the number of rounds.

[1] Each automorphism requires a key-switching operation, which increases the noise in principle but is practically insignificant and is therefore ignored in this work.

3 Design Rationale

In this section we explain and motivate the design decisions made for CHAGHRI in accordance with the discussion in Section 2.

3.1 Motivation of CHAGHRI

CHAGHRI takes *Vision* as a starting point, and improves it with respect to the efficiency metrics specific to BGV rather than ZK-STARK. FHE/BGV-specific efficiency metrics are stated in [36] reasoned by a comparative analysis of *Vision*, *Rescue*, and *AES*.

Our starting point is the comparative analysis performed in [36] for 128-bit security in terms of latency (*i.e.* the time it takes the encryption function to finish).

Benchmarking with a 128-bit state. It is stated in [36] that *AES* performs 88% faster than *Vision*, and 96% faster than *Rescue*. The reason *Vision* and *Rescue* are slower than *AES* is that they require deeper circuits which in turn require a larger cyclotomic polynomial $\Phi(m)$ to evaluate. Therefore, apart from requiring more primitive operations (*i.e.,* multiplications, additions, and automorphisms), the running time of each primitive operation is longer due to the larger $\Phi(m)$.

Benchmarking with larger state sizes. [36] also describes a benchmark for higher throughputs. The motivation behind this is that by increasing the number of state elements in *Vision* its throughput increases while keeping latency constant; whereas for *AES* the increase in throughput forces a linear increase in latency. However, even though *Vision*'s latency asymptotically grows slower than that of *AES* for a higher throughput, the latter still outperforms the former by 45% for a 2048-bit state.

This comparative analysis concludes that the computation of the inversion and the dense affine polynomial are the most expensive operations for larger extensions of the base field. Even though *Vision* and *Rescue* achieve a compact algebraic description in ZK and MPC, they do not seem to perform well in BGV. This is because both *Vision* and *Rescue* make heavy use of ZK and MPC specific non-procedural operations. For instance, inversion is efficiently computed in MPC by means of masking and offloading the heavy operations to the offline phase. However, in FHE this is being unavailable; the number of operations required to compute inversion increases as the degree of the field extension increases. It follows that, different considerations are involved in FHE applications such as circuit depth, suggesting that a novel instance of *Marvellous* may be appropriate.

3.2 Frobenius Automorphism as a Non-procedural Computation

In the variant of the BGV cryptosystem [16] used in this work, converting the polynomial $a(X) \in \mathbb{A}$ to $a^{(i)}(X) \stackrel{\text{def}}{=} a(X^i) \mod \Phi_m(X)$ where m is a cyclotomic polynomial is another primitive operation. When i is in the form p^i, the automorphism is applied to each slot separately and said to be a Frobenius automorphism. The Frobenius

automorphisms increase the noise by a negligible amount when compared to other primitive operations.

We identify Frobenius automorphism as a non-procedural computation since it computes an exponentiation of the form X^{p^i} over $\mathbb{F}_{p^n}$ while the running time of this operation is independent of the exponent.

3.3 Non-linearity

Power maps are widely used in S-box layers owing to their cryptanalytic properties [33]. In *Vision,* inversion is used to construct the S-box. Algorithm 4 describes the pseudo-code for implementing inversion over $\mathbb{F}_{2^n}$. This algorithm requires $\log n$ Frobenius automorphisms and $2(\log n - 1)$ multiplications arranged in a depth-$\log n$ circuit. Therefore, the running time of the inversion operation grows logarithmically as the degree of the field extension increases.

To choose a power map that can be computed more efficiently in FHE, we reviewed some other well studied power maps: $x \mapsto x^{2^k+1}$ (Gold exponents) [33], $x \mapsto x^{2^{2m}-2^m+1}$ (Kasami exponents) [22], and $x \mapsto x^{2^m-2^{m/2}-1}$ (Niho exponents) [11]. In BGV computation of a Gold exponent requires one Frobenius automorphism and one multiplication, a Kasami exponent requires m Frobenius automorphisms and m multiplications, and a Niho exponent requires $m-2$ Frobenius automorphism and $m-2$ multiplications. A Gold exponent is clearly the most efficient power map to be implemented in this setting.

3.4 Affine Polynomials

$\mathbb{F}_2$-linearized affine polynomials (*i.e.* $A(X) = a_0 + \sum_{i=0}^{n-1} a_i \cdot X^{2^i}$ over $\mathbb{F}_{2^n}$) are an efficient way to increase the complexity of the polynomial description of Chaghri because they are efficient to compute with respect to the normal basis. There exists an element $\alpha \in \mathbb{F}_{2^n}$ such that the set $\{\alpha, \alpha^2, ..., \alpha^{2^{n-1}}\}$ constitutes a basis of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$ which is said to be a normal basis. Then, any element $x \in \mathbb{F}_{2^n}$ can be represented as a vector $(a_0, a_1, ..., a_{n-1})$ as follows:

$$x = \sum_{i=0}^{n-1} a_i \alpha^{2^i},$$

such that squaring is simply a right cyclic shift in this representation.

In ZK, to compute x^{2^k}, the intermediate value $x^{2^{k-1}}$ must be computed first, and this motivates the choice of a dense $\mathbb{F}_2$-linearized affine polynomial in *Vision*. However, this is not the case in BGV as x^{2^k} can be directly computed by a Frobenius automorphism. Therefore, a sparse $\mathbb{F}_2$-linearized affine polynomial is a natural choice for CHAGHRI. To ensure that this polynomial is a permutation, we choose coefficients such that the associated Dickson matrix is non-singular [27].

4 Description Of Chaghri

Following the discussion on its design rationale, we now describe CHAGHRI. CHAGHRI is a substitution-permutation (SP) network that has a vector state of three field elements x_0, x_1, x_2. A single CHAGHRI round consists of two identical steps. Each step employs three layers: S-box, linear and subkey injection. The S-box layer applies an S-box π to each of the three state elements, the linear layer is a multiplication of the output vector of the S-box layer by an MDS matrix M of size 3×3, and the subkey injection layer is an XOR operation between the state and the corresponding subkey.

4.1 Primitive Operations

We now describe the primitive operations used in CHAGHRI.

Gold exponent A Gold exponent is employed in the S-box. Conversely to inversion, a Gold exponent can be computed via a single Frobenius automorphism independently of the degree of the field extension. The security properties of Gold exponents were analyzed in [33] where it was shown to be highly non-linear and safe against differential- and linear-cryptanalysis. Nevertheless, their low algebraic degree pose a problem which we mitigate in the same way as *AES* and *Vision* by employing a carefully chosen $\mathbb{F}_2$-linearized affine polynomial.

Since the cost of implementing a Gold exponent x^{2^k+1} is independent of k we would like to maximize its security benefits. For $s = \gcd(k, n)$ where n is the degree of the field extension x^{2^k+1} is a permutation if and only if $\frac{n}{s}$ is odd. Moreover, if n is odd, and co-prime to k the Gold exponent is a differentially 2-uniform permutation which may influence the choice of n. A larger n increases the throughput for a fixed number of elements. As $n = 64$ does not satisfy that $\frac{n}{s}$ is odd we set $n = 63$ and $k = 32$ which in combination with the $\mathbb{F}_2$-linearized affine polynomial provides a high polynomial degree.

S-Box The S-box of a CHAGHRI round, π, is a power map x^α composed with an affine transformation. Following the design rationale explained in Sections 3.3 and 3.4, the S-box π is described as

$$\pi : \ \mathbb{F}_{2^{63}} \mapsto \mathbb{F}_{2^{63}} : \ x \mapsto B(G(x)).$$

where $G : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is given by $G(x) \mapsto x^{2^{32}+1}$ and $B : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is an $\mathbb{F}_2$-linearized affine polynomial whose coefficients are listed in Appendix C. [2]

[2] A preprint version of this paper used $B(x) = \alpha_1 x^8 + \alpha_2$. However, as was shown by Liu *et al.* in [28] this choice of B results in a linear increase of the algebraic degree rather than the desired exponential increase. This in turn enables a practical higher order differential attack. In the same work, Liu *et al.* show that using $B(x) = c'_1 x^{256} + c'_2 x^4 + c'_3 x + c'_4$, results in exponential increasing of the algebraic degree. The coefficients (c'_1, c'_2, c'_3, c'_4) should be selected in a way that B is a permutation. To avoid confusion, the definite version of CHAGHRI is the one published in CCS 2022 after the issue has been fixed.

Table 1: This table describes for each attack (left column) the lower bound we obtained on its complexity (middle column) and derive a safe number of rounds (right column).

Method	Attack complexity	Safe number of rounds
Differential cryptanalysis	$2^{4 \cdot 62N}$	$N \geq 1$
Linear Cryptanalysis	$2^{8 \cdot 31N}$	$N \geq 1$
Higher-order differentials	[28]	$N \geq 4$
Gröbner basis attack	Appendix B	$N \geq 3$

Linear Layer The linear layer diffuses local properties to the entire state. This is realized by a matrix multiplication with an MDS matrix. We follow the same strategy to create our MDS matrices as explained in Section 2.1. Any MDS matrix having the right dimension can be used in the linear layer as the choice of a specific MDS matrix does not contribute to the security arguments.

4.2 Number of Rounds

Let λ be the largest number of rounds that can be attacked using the approaches described in Section 5. Then the safe number of rounds is determined by the following relation:

$$N = 1.5 \max(\lambda, 5),$$

where the constant 5 is a sanity factor suggested by the *Marvellous* designers [3], and 1.5 is a safety margin proposed in [35]. In Table 1, the number of rounds λ that can be attacked using different methods is analyzed.

Concretely, the analysis results in $1.5 \max(\lambda, 5) = 7.5$ and since both steps in a CHAGHRI round are identical, there is no issue with using a non-integral number of rounds. However, for compliance with the Mavellous design strategy we round up and set $N = 8$ as the total number of rounds.

4.3 Decryption

Recalling the *transciphering* example in Section 1, encryption happens on the client side and decryption is realized homomorphically on the server side and is the part we are aiming to optimize. We therefore first describe the decryption algorithm using the BGV-friendly operations we identified, then describe the encryption algorithm using their (heavier) inverses.

CHAGHRI decryption applies the round function 8 times. A key injection takes place before the first round, between every two steps and after the last round. Figure 2 depicts a CHAGHRI round in a decryption flow. The ciphertext and the master key are the inputs to the first round function, and the plaintext is the output of the last round function. Pseudo-code of CHAGHRI decryption function is listed in Algorithm 1.

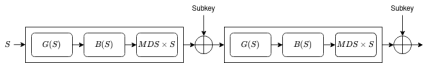

Figure 2: A CHAGHRI decryption round function

Algorithm 1: $Chaghri_{dec}$

Input : Ciphertext C, subkeys K_s for $0 \leq s \leq 2N$
Output: $Chaghri_{dec}(K, C)$
$S_0 = C + K_0$
for $j \leftarrow 1$ ***to*** N **do**
  for $i \leftarrow 0$ ***to*** 2 **do**
    $Inter_j[i] = G(S_{j-1}[i])$
    $Inter_j[i] = B(Inter_j[i])$
  for $i \leftarrow 0$ ***to*** 2 **do**
    $S_j[i] = \sum_{k=0}^{2} M[i,k]Inter_j[k] + K_{2j-1}[i]$
  for $i \leftarrow 0$ ***to*** 2 **do**
    $Inter_j[i] = G(S_j[i])$
    $Inter_j[i] = B(Inter_j[i])$
  for $i \leftarrow 0$ ***to*** 2 **do**
    $S_j[i] = \sum_{k=0}^{2} M[i,k]Inter_j[k] + K_{2j}[i]$
return S_N

4.4 Encryption

The S-box flow in the encryption function is described as

$$\pi : \ \mathbb{F}_{2^{63}} \mapsto \mathbb{F}_{2^{63}} : \ x \mapsto G^{-1}(B^{-1}(x)).$$

where $G^{-1}(x) = x^l$, $l = \dfrac{2^{2048}-1}{2^{64}-1}$ and $B^{-1}(x)$ is given in Appendix C. Figure 3 depicts the a round of the encryption function.

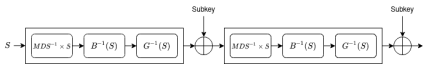

Figure 3: A CHAGHRI encryption round function

4.5 Key Schedule

The key schedule algorithm is indeed the iteratively applied CHAGHRI decryption round function. In order to generate the subkeys, the round function takes the master key instead of the ciphertext input, and takes additional round constants instead of the subkeys injected. The intermediate state after the round constant injection is provided as a subkey. Round constants are used to prevent possible

symmetries and self-similarities in the algorithm to thwart certain type of attacks (*e.g.,* rotational cryptanalysis). The round constants should be chosen such that they are not rotational-invariant and they do not belong to any subfield of $\mathbb{F}_q$. The round constants used in CHAGHRI are given in Appendix D.

5 Security Analysis

In this section we analyze the security of CHAGHRI against applicable statistical and structural attacks. CHAGHRI is secure against statistical attacks due to the large field size and the properties of its Sbox. Each round consists of two polynomial transformations having specific properties to assure the resistance against structural and algebraic attacks. The high degree Gold exponent combined with the affine linearized polynomial ensure that the description is dense and has high degree across all possible polynomial descriptions. In the rest of this section we show that this is indeed the case.

5.1 Statistical Attacks

The most common way to argue the security of a block cipher against differential and linear cryptanalysis is the wide trail strategy [10]. A CHAGHRI round involves two maps: a linearized affine transformation B, and a power map G which is non-linear; whose linear and differential properties were studied in [33]. For an n-bit Boolean function f, the difference propagation probability δ is defined as

$$\delta = 2^{-n} \max_{i,j} \left| \{x | f(x) \oplus f(x \oplus i) = j\} \right|,$$

and the maximum absolute correlation between any pair of linear combinations of n input bits and n output bits is defined as

$$\omega = \max_{\alpha,\beta \in \mathbb{F}_2^n} \left(2 \Pr_{x \in \mathbb{F}_2^n} \left[\alpha x \oplus \beta f(x) = 0 \right] - 1 \right).$$

CHAGHRI has $\delta = 2^{-62}$ and $\omega = 2^{-31}$, see [33] for the details. Since the MDS matrix activates at least $m+1 = 4$ Sboxes in each round (2 steps), the differential transition probability over N rounds is at most

$$2^{(4)(-62)N}$$

and the absolute correlation for any N-round linear trail is at most

$$2^{(4)(-31)N}.$$

The necessary number of rounds to achieve 128 bits of security is $N = 2$ and attacking more rounds of CHAGHRI using statistical attacks is unlikely due to the safety margins we use.

5.2 Structural and Algebraic Attacks

5.2.1 Invariant Subfield Attacks

CHAGHRI operates over the binary field $\mathbb{F}_{2^{63}}$ with subfields of the form $\mathbb{F}_{2^k}$ such that $2^k - 1$ divides $2^{63} - 1$. An example of such subfields are $\mathbb{F}_{2^3}$ or $\mathbb{F}_{2^7}$. CHAGHRI may be vulnerable to the invariant subfield attack [3] if there exists two subfields $\mathbb{F}_{q1} \subset \mathbb{F}_{2^{63}}$ and $\mathbb{F}_{q2} \subset \mathbb{F}_{2^{63}}$ such that for any input to the round function $x \in \mathbb{F}_{q1}$, the corresponding output y lies in $\mathbb{F}_{q2}$. In order to ensure the security of CHAGHRI against this attack, we chose the coefficients of the polynomial B and the constants used in the key schedule so that they do not lie in any subfield of $\mathbb{F}_{2^{63}}$.

5.2.2 Higher-Order Differential Attacks

A higher-order differential attack [24] is an attack targeting the low algebraic degree of transformations used in block ciphers over binary fields. The algebraic degree of a function f is the degree of the monomial with the highest degree when f is given in algebraic normal form. Using the Coefficient Grouping method [28], Chaghri achieves the security of 128 bits in $N = 4$ rounds.

5.2.3 Interpolation Attacks

The interpolation attack [23] targets the low degree of the polynomial description of a block cipher. The attacker reconstructs the polynomial description using plaintext/ciphertext pairs by means of Lagrange interpolation. The complexity of the interpolation is $O(d \log d)$ where d is the degree of the polynomial. In CHAGHRI, the degree of the power map G is $2^{32} + 1$ and the degree of polynomial B is 8. The composition of B and G makes the interpolation attack impractical even after a single round.

5.2.4 Gröbner basis attack

Gröbner basis attacks is believed to be a threatening attack in the literature against algebraic ciphers because their complexity is analogous to the efficiency of the cipher itself. In the Gröbner basis attack, a primitive like a block cipher or a hash function is modeled using a multivariate system of polynomials such that the set of solutions jeopardizes the security claims about the primitive. If the system of polynomials that describes it is easy to solve, the primitive is vulnerable to the attack.

Folklore is that the most successful method in solving a general system of polynomials used in algebraic ciphers is Gröbner basis algorithms such as F4 [15] and F5 [14]. In general, it is hard to compute the exact complexity of the attack for algebraic ciphers. Therefore, we follow the same approach used in [3], and extrapolate the complexity of the attack against the full cipher by running the attack for round reduced versions.
In Appendix B we provide a detailed description of our analysis against the Gröbner basis attack on CHAGHRI. The analysis shows that after three rounds, the complexity of the attack is at least 2^{128}.

5.2.5 Conclusion

we analyzed the security of CHAGHRI against the most promising attacks and found the maximum number of rounds they can break for 128-bit security. In the case of statistical attacks, the largest number of rounds that can be attacked is $N = 1$. We believe other statistical attacks such as truncated differentials attacks [25] or rebound attacks [30, 26] are thwarted by the high safety margins.

In the case of algebraic attacks, the largest number of rounds that can be attacked by higher-order differentials cryptanalysis is $N = 4$. The largest number of rounds that can be attacked using Gröbner basis algorithm is $N = 3$. Other kinds of algebraic attacks do not seem to outperform this. The security against the interpolation attack is achieved by the large polynomial degree and the security against the invariant subfield attack is achieved by a careful choice of constants.

Table 2: A comparison of the running times of *AES* and CHAGHRI. We see that CHAGHRI offers better throughout and a lower running time even when the FHE scheme is using more secure parameters.

Algorithm	State	Throughput (bits)	Cyclotomic Polynomial	FHE Security (bits)	No. Rounds	Running Time (sec.)	Seconds Per Bit
AES	$\mathbb{F}_{2^8}^{16}$	128	Φ_{53261}	141.924	10	97.84	0.76
CHAGHRI	$\mathbb{F}_{2^{63}}^{3}$	189	Φ_{48133}	172.24	8	54.32	0.28

6 Benchmark

To present the benefits of CHAGHRI we implemented it using HElib, a software library implementing the BGV variant described in Section 2.2.1. The library is written in C++ and uses the NTL mathematical library. We compare this to the performance of *AES*-128 using the implementation by Gentry *et al.* [17] that is built into HElib and used in this work without any modification.

6.1 Implementation Details of CHAGHRI

The packed representation of CHAGHRI is as follows:

$$[\alpha_1, \alpha_2, \alpha_3], \qquad \text{where } \alpha_i \in \mathbb{F}_{2^{63}}.$$

The power map of the CHAGHRI S-Box requires one Frobenius automorphism and one multiplication. The $\mathbb{F}_2$-linearized affine polynomial is likewise simple and requires two Frobenius automorphisms and three constant multiplication. The S-Box is implemented via a depth-1.5 circuit and listed in Algorithm 2 and requires one multiplication and three Frobenius automorphisms in total.

The pseudo-code of the linear layer is listed in Algorithm 3.

Subkey injection is simply an addition of the ciphertexts encrypting the state and the subkey.

Algorithm 2: *Chaghri S-Box*

Input : Chaghri state: x

$y = x^{2^{32}}$; `// Frobenius automorphism`

$x = yx$; `// Multiplication (-1 level)`

// Frobenius automorphism + constant multiplication (-0.5 levels)

$sum = c_1 \cdot x^{2^8} + c_2 \cdot x^{2^2} + c_3 \cdot x + c_4$

return sum

Algorithm 3: *Chaghri Matrix Multiplication*

Input : Preprocessed MDS Matrix Rows: MDS, Chaghri state: x

$x' = MDS[1] \cdot (x \gg 1)$; `// Constant multiplication -0.5 levels`

$x'' = MDS[2] \cdot (x \gg 1)$

$x = MDS[0] \cdot x$

return $x + x' + x''$

Expected Cost of a CHAGHRI Round. A Single CHAGHRI round consists of two multiplications, six Frobenius automorphisms and four rotations arranged in a depth-four circuit.

6.2 Evaluation

We benchmarked the *AES* implementation built into HElib, and our CHAGHRI implementation in an environment that runs Ubuntu Server 18.04 LTS with 3 TB RAM and 4 x Intel(R) Xeon(R) Gold 6136 CPU @ 3.00GHz.

Table 2 presents the running times of *AES* and CHAGHRI. **When their running times per bit are compared, CHAGHRI achieves a more compact algebraic description outperforming *AES* by 63%!**[3]

Acknowledgement Tomer Ashur is an FWO post-doctoral fellow under Grant Number 12ZH420N. This work was partially supported by the Research Council KU Leuven, C16/18/004. Additionally, this work was partially done when Dilara Toprakhisar was a Master's student at TU Eindhoven, and was a receiver of ALSP scholarship from TU Eindhoven. Moreover, Dilara Toprakhisar shares her excitement of the first publication with whomever reads the acknowledgements, including her brother.

References

[1] Martin Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for mpc and fhe. Cryptology ePrint Archive, Paper 2016/687, 2016. https://eprint.iacr.org/2016/687.

[3] The preprint version prior to the work of Liu *et al.* claimed 65% here.

[2] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 191–219, 2016.

[3] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols. *IACR Trans. Symmetric Cryptol.*, 2020(3):1–45, 2020.

[4] Luk Bettale, Jean-Charles Faugère, and Ludovic Perret. Solving polynomial systems over finite fields: improved analysis of the hybrid approach. In Joris van der Hoeven and Mark van Hoeij, editors, *International Symposium on Symbolic and Algebraic Computation, ISSAC'12, Grenoble, France - July 22 - 25, 2012*, pages 67–74. ACM, 2012.

[5] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 309–325. ACM, 2012.

[6] Zvika Brakerski and Vinod Vaikuntanathan. Efficient Fully Homomorphic Encryption from (Standard) LWE. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 97–106. IEEE Computer Society, 2011.

[7] 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. Cryptology ePrint Archive, Paper 2015/113, 2015. https://eprint.iacr.org/2015/113.

[8] Carlos Cid, John Petter Indrøy, and Håvard Raddum. Fasta - a stream cipher for fast fhe evaluation. Cryptology ePrint Archive, Paper 2021/1205, 2021. https://eprint.iacr.org/2021/1205.

[9] Orel Cosseron, Clément Hoffmann, Pierrick Méaux, and François-Xavier Standaert. Towards globally optimized hybrid homomorphic encryption - featuring the elisabeth stream cipher. Cryptology ePrint Archive, Paper 2022/180, 2022. https://eprint.iacr.org/2022/180.

[10] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[11] Hans Dobbertin. Almost perfect nonlinear power functions on gf(2n): The niho case. *Inf. Comput.*, 151(1-2):57–72, 1999.

[12] 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. Cryptology ePrint Archive, Paper 2018/181, 2018. https://eprint.iacr.org/2018/181.

[13] Christoph Dobraunig, Lorenzo Grassi, Lukas Helminger, Christian Rechberger, Markus Schofnegger, and Roman Walch. Pasta: A case for hybrid homomorphic encryption. Cryptology ePrint Archive, Paper 2021/731, 2021. https://eprint.iacr.org/2021/731.

[14] Jean Charles Faugère. A new efficient algorithm for computing gröbner bases without reduction to zero (f5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, ISSAC '02, page 75–83, New York, NY, USA, 2002. Association for Computing Machinery.

[15] Jean-Charles Faugére. A new efficient algorithm for computing gröbner bases (f4). *Journal of Pure and Applied Algebra*, 139(1):61–88, 1999.

[16] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2012.

[17] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic Evaluation of the AES Circuit. *IACR Cryptol. ePrint Arch.*, 2012:99, 2012.

[18] Lorenzo Grassi, Daniel Kales, Dmitry Khovratovich, Arnab Roy, Christian Rechberger, and Markus Schofnegger. Starkad and poseidon: New hash functions for zero knowledge proof systems. *IACR Cryptol. ePrint Arch.*, page 458, 2019.

[19] Lorenzo Grassi, Reinhard Lüftenegger, Christian Rechberger, Dragos Rotaru, and Markus Schofnegger. On a generalization of substitution-permutation networks: The HADES design strategy. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 674–704. Springer, 2020.

[20] Jincheol Ha, Seongkwang Kim, Byeonghak Lee, Jooyoung Lee, and Mincheol Son. Rubato: Noisy ciphers for approximate homomorphic encryption (full version). Cryptology ePrint Archive, Paper 2022/537, 2022. https://eprint.iacr.org/2022/537.

[21] Phil Hebborn and Gregor Leander. Dasta - alternative linear layer for rasta. *IACR Trans. Symmetric Cryptol.*, 2020(3):46–86, 2020.

[22] Doreen Hertel. A note on the kasami power function. *IACR Cryptol. ePrint Arch.*, 2005:436, 2005.

[23] Thomas Jakobsen and Lars R. Knudsen. The interpolation attack on block ciphers. In Eli Biham, editor, *Fast Software Encryption*, pages 28–40, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[24] Lars R. Knudsen. Truncated and higher order differentials. In Bart Preneel, editor, *Fast Software Encryption*, pages 196–211, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[25] Lars Ramkilde Knudsen. Truncated and higher order differentials. In *FSE*, 1994.

[26] Mario Lamberger, Florian Mendel, Christian Rechberger, Vincent Rijmen, and Martin Schläffer. Rebound distinguishers: Results on the full whirlpool compression function. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 126–143. Springer, 2009.

[27] Rudolf Lidl and Harald Niederreiter. *Finite fields*. Number 20. Cambridge university press, 1997.

[28] Fukang Liu, Ravi Anand, Libo Wang, Willi Meier, and Takanori Isobe. Coefficient grouping: Breaking chaghri and more. Cryptology ePrint Archive, Paper 2022/991, 2022. https://eprint.iacr.org/2022/991.

[29] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic Methods for Interactive Proof Systems. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume I*, pages 2–10. IEEE Computer Society, 1990.

[30] Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. The rebound attack: Cryptanalysis of reduced whirlpool and grøstl. In *FSE*, 2009.

[31] Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. Towards stream ciphers for efficient fhe with low-noise ciphertexts. Cryptology ePrint Archive, Paper 2016/254, 2016. https://eprint.iacr.org/2016/254.

[32] Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In Christian Cachin and Thomas Ristenpart, editors, *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011*, pages 113–124. ACM, 2011.

[33] Kaisa Nyberg. Differentially Uniform Mappings for Cryptography. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 55–64. Springer, 1993.

[34] Nigel P. Smart and Frederik Vercauteren. Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.

[35] Alan Szepieniec, Tomer Ashur, and Siemen Dhooghe. Rescue-prime: a standard specification (sok). *IACR Cryptol. ePrint Arch.*, page 1143, 2020.

[36] Dilara Toprakhisar, Mohammad Mahzoun, and Tomer Ashur. A Comparative Study of Vision and AES in FHE Setting. August 2021. The Conference for Failed Approaches and Insightful Losses in Cryptology, CFail; Conference date: 14-08-2021.

[37] Baofeng Wu and Zhuojun Liu. Linearized polynomials over finite fields revisited. *Finite Fields Their Appl.*, 22:79–100, 2013.

A Inversion Algorithm in AES implementation

For completeness, we describe the *AES* inversion algorithm in algorithm 4.

Algorithm 4: *Inversion over* $\mathbb{F}_{2^n}$

Input : x, n
$exp \leftarrow 2$
$y \leftarrow x^2$; // Frobenius automorphism
$z \leftarrow x^2 \cdot x = x^3$; // Multiplication (-1 level)
for $i \leftarrow 2$ ***to*** $\log n$ **do**
  $x = (z)^{2^{exp}}$; // Frobenius automorphism
  $y = x \cdot y$; // Multiplication (-1 level)
  if $i == \log n$ **then**
    return y
  end
  $z = x \cdot z$; // Multiplication
  $exp = 2 \cdot exp$
end

B Detailed GB analysis

We first describe the details of modeling the attack, then we present the results obtained after running it. For the implementation We used Magma on a Linux machine with 54GB of RAM and Intel(R) Xeon(R) CPU E3-1275 v6 running at 3.80GHz.

To argue the security of CHAGHRI we employ the approach offered by the Marvellous design strategy as follows:

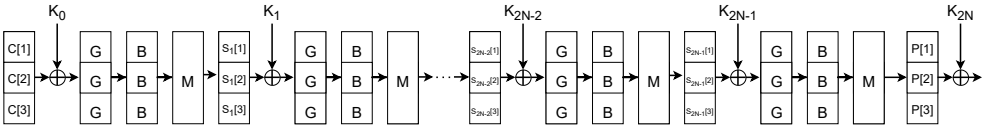

Figure 4: Decryption in Chaghri. $P \oplus K_{2N}$ is the plaintext, S_i's are the intermediate states and C is the ciphertext.

- First, model CHAGHRI as a system of multivariate polynomials;
- Then, compute the Gröbner basis in *degrevlex* order. For this we use the F4 algorithm offered in Magma.

By the *Marvellous* design strategy, the algorithm is designed such that computing the Gröbner basis in degrevlex order should already be prohibitively expensive.

The cost is estimated using empirical data of reduced versions that is extrapolated to the complete parameter set. In general, the complexity of computing the Gröbner [4] basis is

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

Where n is the number of variables in the polynomial system, $2 < \omega < 3$ is the linear algebra constant, and d_{reg} is the degree of the regularity of the system.

The attack details. We model the block cipher as depicted in Figure 4.

Let us denote the number of rounds by N, the dimension by m, and the internal states by $S_i : 1 \le i \le 2N-1$. The round keys are denoted by $K_i : 0 \le i \le 2N$. Finally, variables $\mathscr{C}_i : 0 \le i \le 2N$ are the round constants. The system in Figure 5 describes CHAGHRI.

In Figure 5, the first three lines are the equations modeling the encryption of P using K resulting in C. The last line describes the equations of round keys. There are in total $4mN$ equations in $4mN$ variables. Variables are $\{S_i[j];\ 1 < i < 2N\,, j \le m\} \cup \{K_i[j];\ 0 \le i \le 2N\,, j \le m\}$.

One of the main challenges in modeling CHAGHRI is modeling the Gold exponent $G = 2^{2^{32}+1}$, which is of a high degree. To overcome this challenge, we try to model the polynomial F using another representation having smaller degree.

For t a primitive element of $\mathbb{F}_{2^{63}}$, $P(t)$ an arbitrary polynomial in t, Lemma B.1 shows that $G(P(t)) = P(\mathscr{G}(t)) \times P(t)$. This implies that if finding a low degree transformation $\mathscr{F}(t)$ allows to model G using a low degree polynomial and solving the system more easily.

The following equality is derived from a sage program that computes the result of

$$\left\{\begin{array}{ll} B(G(C[\beta]\oplus K_0[\beta]))\oplus\sum_{\alpha=1}^{m} M^{-1}[\beta,\alpha](S_1[\alpha]\oplus K_1[\alpha])=0 & \beta\in[m] \\ B(G(S_{i-1}[\beta]))\oplus\sum_{\alpha=1}^{m} M^{-1}[\beta,\alpha](S_i[\alpha]\oplus K_i[\alpha])=0 & 1<i<2N,\ \beta\in[m] \\ B(G(S_{2N-1}[\beta]))\oplus\sum_{\alpha=1}^{m} M^{-1}[\beta,\alpha](P[\alpha]\oplus K_{2N}[\alpha])=0 & \beta\in[m] \\ K_i[\beta]\oplus\mathscr{C}_i[\beta]\oplus\sum_{\alpha=1}^{m} M[\beta,\alpha]\cdot B(G(K_{i-1}))=0 & 1\le i\le 2N\ \beta\in[m] \end{array}\right\}$$

Figure 5: The system of the polynomial equations modeling the CHAGHRI block cipher. $K_i: 0\le i\le 2N$ are the variables for the round keys, P and C are the variables for the plaintext and the ciphertext, and $S_i: 0<i<2N$ are the intermediate variables; $\mathscr{C}_i: 0\le i\le 2N$ are the round constants.

$\mathscr{F}(t)$ as a Frobenius automorphism.

$$\begin{aligned}\mathscr{F}(t) = {} & t^{59}+t^{58}+t^{57}+t^{55}+t^{53}+t^{52}+t^{51}+t^{46}+t^{45}+t^{44}\\ & +t^{43}+t^{42}+t^{41}+t^{39}+t^{37}+t^{34}+t^{30}+t^{29}+t^{28}+t^{27}+t^{26}\\ & +t^{25}+t^{23}+t^{22}+t^{19}+t^{17}+t^{14}+t^{13}+t^{10}+t^{8}+t^{6}+t^{5}+t^{3}\\ & +t^{2}+t\end{aligned}$$

Therefore, we model $G(t)$ as $\mathscr{F}(t)\times t$ which is a polynomial of degree 60 instead of degree $2^{32}+1$.

Lemma B.1. *Let $\mathscr{P}$ be a polynomial with binary coefficients and let $F(t)=t^{2^{32}}$ be the Frobenius transformation on $\mathbb{F}_{2^{63}}$. Then $\forall t\in\mathbb{F}_{2^{63}}: F(\mathscr{P}(t))=\mathscr{P}(F(t))$.*

Proof. For $b_i\in\{0,1\}$ let $\mathscr{P}(t)=\sum_{i=0}^{n} b_i t^i$. Since the characteristic of the field $\mathbb{F}_{2^{63}}$ is 2, then $(a+b)^2=a^2+b^2$. We obtain the following:

$$F(\mathscr{P}(t)) = (\mathscr{P}(t))^{2^{32}} \tag{2}$$

$$\Rightarrow F(\mathscr{P}(t)) = (\sum_{i=0}^{n} b_i t^i)^{2^{32}} \tag{3}$$

$$\Rightarrow F(\mathscr{P}(t)) = \sum_{i=0}^{n} b_i t^{i(2^{32})} \tag{4}$$

$$\Rightarrow F(\mathscr{P}(t)) = \mathscr{P}(F(t)) \tag{5}$$

□

The next step is to compute the degree of regularity of the system in order to estimate the complexity of the attack which is:

$$O\left(\binom{4mN + d_{reg}}{d_{reg}}^{\omega}\right).$$

To extrapolate the degree of the regularity, we run the Gröbner basis algorithm and compute the degree of regularity for a small number of rounds.

Our environment was not strong enough to support the computation and crashed before halting. However, before that it reached a step degree of 220, which can be used to lower bound the degree of regularity. The other observation is that the degree of regularity grows linearly with the number of rounds. Using the lower bound for the degree of regularity, we can compute the complexity of the Gröbner basis attack for N rounds using the following relation.

$$\binom{4mN + d_{con.lb}}{d_{con.lb}}^2$$

The number of rounds that can be attacked is calculated as

$$l_1 = \min(N) \quad \text{subject to} \quad \binom{4mN + d_{con}}{d_{con}}^2 \geq 2^s,$$

and achieves 128 bits of security when $N = 3$.

C The Polynomial $B(x)$ and $B^{-1}(x)$

The $\mathbb{F}_2$-linearized affine polynomial used in Chaghri is:

$$B(x) = c_1 x^{256} + c_2 x^4 + c_3 x + c_4$$

where

$c_1 = t^{60} + t^{59} + t^{58} + t^{56} + t^{53} + t^{52} + t^{50} + t^{48} + t^{46} + t^{44} + t^{43} + t^{40} + t^{39} + t^{38} + t^{37} + t^{35} + t^{33} + t^{31} + t^{28} + t^{27} + t^{26} + t^{23} + t^{21} + t^{20} + t^{18} + t^{16} + t^{15} + t^{14} + t^{12} + t^{10} + t^9 + t^6 + t^5 + t^2 + t$

$c_2 = t^{62} + t^{61} + t^{60} + t^{57} + t^{56} + t^{51} + t^{50} + t^{49} + t^{48} + t^{46} + t^{45} + t^{44} + t^{43} + t^{40} + t^{37} + t^{33} + t^{32} + t^{31} + t^{30} + t^{29} + t^{25} + t^{22} + t^{20} + t^{18} + t^{17} + t^{15} + t^{14} + t^{12} + t^{11} + t^{10} + t^5 + t + 1$

$c_3 = t^{61} + t^{59} + t^{57} + t^{54} + t^{53} + t^{52} + t^{50} + t^{49} + t^{48} + t^{47} + t^{45} + t^{42} + t^{40} + t^{37} + t^{36} + t^{35} + t^{34} + t^{33} + t^{31} + t^{29} + t^{26} + t^{24} + t^{23} + t^{21} + t^{20} + t^{19} + t^{17} + t^{14} + t^{13} + t^8 + t^6 + t^4 + 1$

$c_4 = t^{59} + t^{55} + t^{54} + t^{53} + t^{52} + t^{50} + t^{49} + t^{46} + t^{45} + t^{42} + t^{40} + t^{38} + t^{31} + t^{30} + t^{26} + t^{24} + t^{23} + t^{22} + t^{19} + t^{15} + t^{13} + t^{12} + t^7 + t^2 + t$

The polynomial B^{-1} has the form

$$B^{-1}(x) = \sum_{i=0}^{62} c_i x^{2^i}$$

where c_i is the i^{th} element in the first row of adjugate matrix of Dickson matrix associated to B [37]. The code to generate the coefficients of B^{-1} is available at GitHub[4].

The coefficients of the polynomial $B(x) = \alpha_1 x^8 + \alpha_2$ used in the preprint version of Chaghri is as follow:

$$\alpha_1 = t^{61} + t^{57} + t^{56} + t^{55} + t^{54} + t^{52} + t^{50} + t^{49} + t^{45} + t^{44} + t^{41} + t^{37} + t^{34} + t^{32} + t^{31} + t^{30} + t^{29} + t^{27} + t^{26} + t^{25} + t^{24} + t^{23} + t^{22} + t^{19} + t^{16} + t^{12} + t^{11} + t^{10} + t^8 + t^6 + t^5 + t^4 + t^3 + 1$$

$$\alpha_2 = t\,t^{60} + t^{57} + t^{52} + t^{47} + t^{44} + t^{41} + t^{39} + t^{37} + t^{35} + t^{34} + t^{31} + t^{30} + t^{29} + t^{28} + t^{24} + t^{23} + t^{21} + t^{20} + t^{19} + t^{18} + t^{14} + t^{13} + t^{11} + t^{10} + t^8 + t^6 + t^5 + t^3 + t^2 + 1$$

D Round Constants

Round 1:

$$t^{61} + t^{60} + t^{57} + t^{56} + t^{55} + t^{54} + t^{51} + t^{49} + t^{47} + t^{45} + t^{44} + t^{43} + t^{41} + t^{39} + t^{36} + t^{35} + t^{34} + t^{33} + t^{29} + t^{27} + t^{26} + t^{25} + t^{23} + t^{21} + t^{16} + t^{14} + t^{12} + t^8 + t^3 + t^2 + t + 1,$$

$$t^{62} + t^{59} + t^{54} + t^{53} + t^{52} + t^{51} + t^{49} + t^{47} + t^{45} + t^{44} + t^{42} + t^{41} + t^{38} + t^{37} + t^{35} + t^{34} + t^{32} + t^{31} + t^{30} + t^{29} + t^{28} + t^{22} + t^{21} + t^{19} + t^{14} + t^{12} + t^{10} + t^8 + t^7 + t^6 + t^3 + 1,$$

$$t^{61} + t^{60} + t^{57} + t^{56} + t^{52} + t^{51} + t^{46} + t^{44} + t^{43} + t^{42} + t^{41} + t^{37} + t^{35} + t^{31} + t^{28} + t^{26} + t^{22} + t^{21} + t^{19} + t^{15} + t^2 + t + 1;$$

$$t^{56} + t^{54} + t^{52} + t^{51} + t^{48} + t^{45} + t^{41} + t^{39} + t^{36} + t^{33} + t^{28} + t^{27} + t^{25} + t^{23} + t^{19} + t^{15} + t^{13} + t^{12} + t^{11} + t^9 + t^8 + t^6 + t^4 + t^2 + t,$$

$$t^{62} + t^{61} + t^{58} + t^{54} + t^{51} + t^{50} + t^{49} + t^{47} + t^{46} + t^{42} + t^{41} + t^{40} + t^{39} + t^{34} + t^{33} + t^{32} + t^{27} + t^{25} + t^{24} + t^{23} + t^{21} + t^{19} + t^{17} + t^{16} + t^{14} + t^8 + t^7 + t^5 + t + 1,$$

$$t^{62} + t^{61} + t^{58} + t^{54} + t^{53} + t^{52} + t^{49} + t^{44} + t^{39} + t^{37} + t^{35} + t^{33} + t^{32} + t^{29} + t^{27} + t^{21} + t^{20} + t^{18} + t^{17} + t^{16} + t^{13} + t^{12} + t^{10} + t^9 + t^8 + t^7 + t^5 + t^3 + t + 1;$$

Round 2:

$$t^{61} + t^{60} + t^{59} + t^{58} + t^{57} + t^{54} + t^{53} + t^{51} + t^{50} + t^{49} + t^{47} + t^{45} + t^{44} + t^{40} + t^{39} + t^{38} + t^{36} + t^{32} + t^{30} + t^{28} + t^{27} + t^{26} + t^{25} + t^{22} + t^{21} + t^{20} + t^{19} + t^{17} + t^{16} + t^{14} + t^{13} + t^9 + t^7 + t^6 + t^5 + t^4 + t,$$

$$t^{62} + t^{60} + t^{59} + t^{58} + t^{57} + t^{56} + t^{55} + t^{54} + t^{53} + t^{50} + t^{47} + t^{45} + t^{44} + t^{41} + t^{40} + t^{39} + t^{38} + t^{37} + t^{35} + t^{31} + t^{28} + t^{26} + t^{25} + t^{24} + t^{22} + t^{19} + t^3 + 1,$$

[4] https://gist.github.com/mahzoun/915b13826c23a4450ec266a889106bfb

$t^{61}+t^{59}+t^{56}+t^{53}+t^{52}+t^{49}+t^{48}+t^{47}+t^{44}+t^{43}+t^{42}+t^{40}+t^{39}+t^{38}+t^{36}+t^{34}+t^{33}+$
$t^{32}+t^{31}+t^{30}+t^{29}+t^{26}+t^{25}+t^{24}+t^{23}+t^{22}+t^{21}+t^{18}+t^{16}+t^{14}+t^{12}+t^{10}+t^{8}+t^{6}+t^{5}+1;$

$t^{61}+t^{59}+t^{54}+t^{51}+t^{50}+t^{48}+t^{47}+t^{46}+t^{45}+t^{44}+t^{42}+t^{40}+t^{39}+t^{38}+t^{37}+t^{36}+$
$t^{33}+t^{31}+t^{26}+t^{24}+t^{13}+t^{11}+t^{6}+t+1,$

$t^{62}+t^{61}+t^{60}+t^{58}+t^{52}+t^{51}+t^{49}+t^{48}+t^{46}+t^{45}+t^{43}+t^{41}+t^{40}+t^{39}+t^{36}+t^{32}+$
$t^{28}+t^{19}+t^{17}+t^{16}+t^{15}+t^{14}+t^{10}+t^{8}+t^{6}+t^{2}+1,$

$t^{62}+t^{61}+t^{60}+t^{57}+t^{56}+t^{54}+t^{53}+t^{52}+t^{47}+t^{45}+t^{38}+t^{37}+t^{35}+t^{34}+t^{33}+t^{30}+$
$t^{29}+t^{23}+t^{19}+t^{12}+t^{11}+t^{8}+t^{6}+t^{4}+t^{2}+1;$

Round 3:

$t^{62}+t^{61}+t^{60}+t^{53}+t^{52}+t^{51}+t^{48}+t^{46}+t^{43}+t^{42}+t^{39}+t^{38}+t^{36}+t^{29}+t^{27}+t^{23}+$
$t^{22}+t^{19}+t^{17}+t^{16}+t^{15}+t^{14}+t^{11}+t^{8}+t^{6}+t+1,$

$t^{60}+t^{58}+t^{57}+t^{54}+t^{52}+t^{50}+t^{46}+t^{45}+t^{42}+t^{40}+t^{38}+t^{37}+t^{36}+t^{35}+t^{34}+t^{33}+$
$t^{29}+t^{27}+t^{25}+t^{23}+t^{22}+t^{20}+t^{17}+t^{16}+t^{15}+t^{14}+t^{10}+t^{5}+t^{3}+t,$

$t^{62}+t^{61}+t^{58}+t^{57}+t^{56}+t^{55}+t^{48}+t^{47}+t^{45}+t^{43}+t^{41}+t^{39}+t^{37}+t^{34}+t^{30}+t^{29}+$
$t^{25}+t^{24}+t^{23}+t^{21}+t^{19}+t^{16}+t^{14}+t^{13}+t^{10}+t^{9}+t^{8}+t^{3}+1;$

$t^{62}+t^{61}+t^{60}+t^{59}+t^{58}+t^{57}+t^{53}+t^{52}+t^{50}+t^{49}+t^{48}+t^{45}+t^{43}+t^{42}+t^{41}+t^{37}+$
$t^{36}+t^{35}+t^{34}+t^{31}+t^{30}+t^{29}+t^{28}+t^{27}+t^{26}+t^{24}+t^{23}+t^{21}+t^{19}+t^{18}+t^{14}+t^{10}+t^{4},$

$t^{62}+t^{57}+t^{54}+t^{52}+t^{51}+t^{50}+t^{47}+t^{44}+t^{43}+t^{40}+t^{37}+t^{36}+t^{35}+t^{33}+t^{25}+t^{23}+$
$t^{20}+t^{19}+t^{18}+t^{12}+t^{8}+t^{6}+t^{5}+t^{2},$

$t^{60}+t^{58}+t^{57}+t^{56}+t^{54}+t^{49}+t^{48}+t^{47}+t^{45}+t^{41}+t^{39}+t^{38}+t^{37}+t^{34}+t^{31}+t^{27}+$
$t^{25}+t^{24}+t^{21}+t^{20}+t^{18}+t^{13}+t^{9}+t^{8}+t^{6}+t^{3}+t+1;$

Round 4:

$t^{62}+t^{61}+t^{59}+t^{52}+t^{48}+t^{45}+t^{43}+t^{40}+t^{35}+t^{34}+t^{29}+t^{26}+t^{23}+t^{20}+t^{19}+t^{18}+$
$t^{17}+t^{15}+t^{14}+t^{11}+t^{9}+t^{8}+t^{5}+t^{4}+t^{3},$

$t^{62}+t^{61}+t^{60}+t^{56}+t^{55}+t^{53}+t^{52}+t^{49}+t^{48}+t^{47}+t^{41}+t^{38}+t^{35}+t^{34}+t^{32}+t^{31}+$
$t^{24}+t^{23}+t^{20}+t^{19}+t^{17}+t^{16}+t^{15}+t^{14}+t^{11}+t^{8}+t^{5}+t^{3}+t^{2},$

$t^{61}+t^{60}+t^{58}+t^{57}+t^{56}+t^{55}+t^{53}+t^{50}+t^{49}+t^{44}+t^{39}+t^{37}+t^{36}+t^{35}+t^{34}+t^{31}+$
$t^{30}+t^{25}+t^{24}+t^{23}+t^{19}+t^{18}+t^{15}+t^{14}+t^{13}+t^{7}+t^{5}+t^{3}+t^{2}+1;$

$t^{60}+t^{58}+t^{57}+t^{56}+t^{52}+t^{50}+t^{49}+t^{44}+t^{40}+t^{39}+t^{37}+t^{35}+t^{32}+t^{31}+t^{30}+t^{26}+$
$t^{23}+t^{20}+t^{19}+t^{18}+t^{17}+t^{16}+t^{15}+t^{14}+t^{11}+t^{10}+t^{8}+t^{4}+t,$

$t^{61}+t^{60}+t^{59}+t^{58}+t^{50}+t^{43}+t^{41}+t^{38}+t^{37}+t^{34}+t^{32}+t^{31}+t^{28}+t^{26}+t^{25}+t^{24}+t^{23}+t^{22}+t^{19}+t^{17}+t^{14}+t^{12}+t^{5}+t,$

$t^{62}+t^{61}+t^{58}+t^{57}+t^{51}+t^{50}+t^{48}+t^{46}+t^{44}+t^{42}+t^{41}+t^{40}+t^{35}+t^{32}+t^{26}+t^{25}+t^{21}+t^{18}+t^{17}+t^{16}+t^{15}+t^{14}+t^{12}+t^{10}+t^{8}+t^{5}+t^{2};$

Round 5:

$t^{62}+t^{58}+t^{57}+t^{54}+t^{53}+t^{52}+t^{50}+t^{49}+t^{48}+t^{46}+t^{40}+t^{38}+t^{37}+t^{35}+t^{34}+t^{33}+t^{28}+t^{27}+t^{26}+t^{25}+t^{24}+t^{23}+t^{22}+t^{21}+t^{18}+t^{15}+t^{14}+t^{12}+t^{11}+t^{10}+t^{9}+t^{8}+t^{7}+t^{6}+t^{5}+t+1,$

$t^{62}+t^{61}+t^{60}+t^{59}+t^{58}+t^{56}+t^{54}+t^{53}+t^{51}+t^{49}+t^{48}+t^{47}+t^{42}+t^{41}+t^{39}+t^{37}+t^{36}+t^{35}+t^{33}+t^{32}+t^{26}+t^{24}+t^{22}+t^{20}+t^{19}+t^{18}+t^{14}+t^{13}+t^{9}+t^{8}+t^{7}+t^{5}+t^{2}+1,$

$t^{61}+t^{60}+t^{56}+t^{55}+t^{54}+t^{53}+t^{52}+t^{51}+t^{47}+t^{42}+t^{41}+t^{40}+t^{37}+t^{36}+t^{35}+t^{33}+t^{29}+t^{28}+t^{27}+t^{24}+t^{23}+t^{21}+t^{20}+t^{18}+t^{16}+t^{15}+t^{12}+t^{11}+t^{8}+t^{6}+t^{4}+t^{2}+t+1;$

$t^{62}+t^{60}+t^{57}+t^{56}+t^{55}+t^{54}+t^{53}+t^{49}+t^{48}+t^{46}+t^{45}+t^{42}+t^{40}+t^{39}+t^{34}+t^{33}+t^{31}+t^{25}+t^{24}+t^{23}+t^{21}+t^{19}+t^{17}+t^{14}+t^{13}+t^{11}+t^{10}+t^{9}+t^{4},$

$t^{61}+t^{59}+t^{58}+t^{57}+t^{56}+t^{54}+t^{53}+t^{51}+t^{50}+t^{46}+t^{42}+t^{41}+t^{40}+t^{39}+t^{38}+t^{37}+t^{35}+t^{34}+t^{32}+t^{31}+t^{29}+t^{27}+t^{26}+t^{25}+t^{22}+t^{20}+t^{19}+t^{17}+t^{13}+t^{5}+t^{3}+t^{2}+t+1,$

$t^{62}+t^{53}+t^{50}+t^{48}+t^{47}+t^{46}+t^{44}+t^{42}+t^{41}+t^{40}+t^{38}+t^{36}+t^{34}+t^{25}+t^{24}+t^{22}+t^{20}+t^{19}+t^{18}+t^{16}+t^{11}+t^{10}+t^{9}+t^{8}+t^{7}+t^{5}+t^{3}+t;$

Round 6:

$t^{62}+t^{59}+t^{57}+t^{54}+t^{51}+t^{47}+t^{46}+t^{44}+t^{43}+t^{41}+t^{40}+t^{39}+t^{38}+t^{36}+t^{31}+t^{29}+t^{27}+t^{26}+t^{25}+t^{23}+t^{22}+t^{21}+t^{19}+t^{15}+t^{13}+t^{10}+t^{9}+t^{8}+t^{6}+t^{5}+t^{4}+t^{3}+t^{2}+t,$

$t^{61}+t^{60}+t^{59}+t^{52}+t^{49}+t^{48}+t^{42}+t^{40}+t^{36}+t^{31}+t^{22}+t^{21}+t^{20}+t^{19}+t^{18}+t^{16}+t^{15}+t^{14}+t^{12}+t^{10}+t^{9}+t^{4}+t^{2}+t+1,$

$t^{60}+t^{59}+t^{56}+t^{53}+t^{50}+t^{47}+t^{46}+t^{42}+t^{41}+t^{36}+t^{35}+t^{28}+t^{26}+t^{19}+t^{15}+t^{13}+t^{12}+t^{11}+t^{10}+t^{8}+t^{5}+t^{2};$

$t^{62}+t^{59}+t^{58}+t^{56}+t^{55}+t^{54}+t^{51}+t^{46}+t^{43}+t^{42}+t^{40}+t^{39}+t^{38}+t^{37}+t^{33}+t^{31}+t^{28}+t^{27}+t^{25}+t^{24}+t^{23}+t^{22}+t^{21}+t^{20}+t^{19}+t^{17}+t^{16}+t^{14}+t^{13}+t^{12}+t^{7}+t^{6}+t^{4}+1,$

$t^{59}+t^{57}+t^{54}+t^{49}+t^{47}+t^{46}+t^{45}+t^{44}+t^{41}+t^{29}+t^{26}+t^{22}+t^{13}+t^{11}+t^{9}+t^{7}+t^{6}+t^{5}+t^{4}+t^{3}+t^{2}+t+1,$

$t^{62}+t^{61}+t^{60}+t^{59}+t^{58}+t^{55}+t^{54}+t^{53}+t^{49}+t^{43}+t^{36}+t^{35}+t^{34}+t^{33}+t^{32}+t^{31}+t^{30}+t^{28}+t^{26}+t^{25}+t^{24}+t^{21}+t^{18}+t^{15}+t^{12}+t^{11}+t^{10}+t^{8}+t^{7}+t^{6}+t^{5}+t^{3}+t+1;$

Round 7:

$$t^{62}+t^{61}+t^{57}+t^{55}+t^{51}+t^{50}+t^{49}+t^{48}+t^{47}+t^{46}+t^{45}+t^{44}+t^{39}+t^{38}+t^{37}+t^{36}+t^{35}+t^{34}+t^{33}+t^{32}+t^{30}+t^{27}+t^{24}+t^{23}+t^{22}+t^{19}+t^{17}+t^{15}+t^{14}+t^{12}+t^{10}+t^{7}+t^{6}+t^{5}+t^{3},$$

$$t^{62}+t^{60}+t^{59}+t^{58}+t^{57}+t^{55}+t^{54}+t^{52}+t^{50}+t^{47}+t^{45}+t^{44}+t^{41}+t^{39}+t^{38}+t^{36}+t^{35}+t^{31}+t^{29}+t^{28}+t^{23}+t^{22}+t^{19}+t^{17}+t^{16}+t^{15}+t^{14}+t^{12}+t^{9}+t^{8}+t^{4}+t^{2}+1,$$

$$t^{61}+t^{60}+t^{56}+t^{55}+t^{52}+t^{50}+t^{49}+t^{46}+t^{44}+t^{43}+t^{40}+t^{39}+t^{38}+t^{37}+t^{36}+t^{34}+t^{32}+t^{31}+t^{28}+t^{23}+t^{22}+t^{21}+t^{18}+t^{17}+t^{15}+t^{14}+t^{12}+t^{11}+t^{10}+t^{9}+t^{7}+t^{6}+t^{4}+t^{3}+t^{2}+1;$$

$$t^{59}+t^{57}+t^{56}+t^{55}+t^{54}+t^{53}+t^{52}+t^{49}+t^{47}+t^{45}+t^{41}+t^{39}+t^{38}+t^{37}+t^{35}+t^{34}+t^{31}+t^{30}+t^{26}+t^{25}+t^{21}+t^{20}+t^{19}+t^{18}+t^{11}+t^{10}+t^{9}+t^{8}+t^{7},$$

$$t^{62}+t^{60}+t^{58}+t^{57}+t^{56}+t^{54}+t^{51}+t^{47}+t^{43}+t^{42}+t^{41}+t^{38}+t^{36}+t^{32}+t^{31}+t^{29}+t^{28}+t^{25}+t^{22}+t^{15}+t^{14}+t^{7}+t^{6}+t^{5}+t^{4}+t^{3}+t,$$

$$t^{61}+t^{58}+t^{57}+t^{56}+t^{55}+t^{54}+t^{52}+t^{51}+t^{49}+t^{48}+t^{46}+t^{37}+t^{31}+t^{27}+t^{26}+t^{23}+t^{22}+t^{20}+t^{19}+t^{17}+t^{14}+t^{13}+t^{9}+t^{8}+t^{7}+t^{6}+t^{4}+t+1;$$

Round 8:

$$t^{61}+t^{58}+t^{45}+t^{41}+t^{36}+t^{34}+t^{33}+t^{30}+t^{29}+t^{27}+t^{26}+t^{25}+t^{24}+t^{19}+t^{18}+t^{16}+t^{15}+t^{14}+t^{11}+t^{10}+t^{8}+t^{6}+t^{4}+1,$$

$$t^{62}+t^{61}+t^{59}+t^{57}+t^{56}+t^{55}+t^{53}+t^{46}+t^{42}+t^{40}+t^{39}+t^{38}+t^{34}+t^{32}+t^{29}+t^{28}+t^{27}+t^{24}+t^{21}+t^{20}+t^{19}+t^{18}+t^{17}+t^{16}+t^{15}+t^{12}+t^{11}+t^{10}+t^{9}+t^{8}+t^{5}+t^{4}+t^{3}+t^{2},$$

$$t^{62}+t^{61}+t^{60}+t^{56}+t^{52}+t^{49}+t^{48}+t^{47}+t^{46}+t^{44}+t^{43}+t^{40}+t^{39}+t^{37}+t^{33}+t^{32}+t^{28}+t^{26}+t^{25}+t^{24}+t^{23}+t^{20}+t^{16}+t^{15}+t^{11}+t^{9}+t^{7}+t^{6}+t^{5}+t^{2}+t+1;$$

$$t^{58}+t^{57}+t^{56}+t^{54}+t^{53}+t^{52}+t^{51}+t^{47}+t^{46}+t^{43}+t^{41}+t^{37}+t^{36}+t^{34}+t^{30}+t^{29}+t^{28}+t^{27}+t^{26}+t^{25}+t^{24}+t^{23}+t^{19}+t^{17}+t^{15}+t^{14}+t^{11}+t^{9}+t^{2}+t,$$

$$t^{62}+t^{59}+t^{56}+t^{54}+t^{53}+t^{50}+t^{49}+t^{48}+t^{47}+t^{46}+t^{45}+t^{44}+t^{43}+t^{39}+t^{38}+t^{35}+t^{34}+t^{33}+t^{31}+t^{30}+t^{29}+t^{28}+t^{20}+t^{18}+t^{14}+t^{11}+t^{8}+t^{2}+t+1,$$

$$t^{61}+t^{60}+t^{59}+t^{58}+t^{57}+t^{55}+t^{51}+t^{48}+t^{47}+t^{43}+t^{39}+t^{38}+t^{37}+t^{34}+t^{33}+t^{28}+t^{24}+t^{22}+t^{21}+t^{20}+t^{19}+t^{18}+t^{17}+t^{14}+t^{13}+t^{11}+t^{9}+t^{8}+t^{7}+t^{6}+t^{4}+t^{3}+t+1.$$