Cryptanalysis of Elisabeth-4

Henri Gilbert[1,2], Rachelle Heim Boissier[2], Jérémy Jean[1], and Jean-René Reinhard[1]

[1] ANSSI, Paris, France
[2] Université Paris-Saclay, UVSQ, CNRS,
Laboratoire de mathématiques de Versailles, Versailles, France
rachelle.heim@uvsq.fr

Abstract. `Elisabeth-4` is a stream cipher tailored for usage in hybrid homomorphic encryption applications that has been introduced by Cosseron et al. at ASIACRYPT 2022. In this paper, we present several variants of a key-recovery attack on the full `Elisabeth-4` that break the 128-bit security claim of that cipher. Our most optimized attack is a chosen-IV attack with a time complexity of 2^{88} elementary operations, a memory complexity of 2^{54} bits and a data complexity of 2^{41} bits.
Our attack applies the linearization technique to a nonlinear system of equations relating some keystream bits to the key bits and exploits specificities of the cipher to solve the resulting linear system efficiently. First, due to the structure of the cipher, the system to solve happens to be very sparse, which enables to rely on sparse linear algebra and most notably on the Block Wiedemann algorithm. Secondly, the algebraic properties of the two nonlinear ingredients of the filtering function cause rank defects which can be leveraged to solve the linearized system more efficiently with a decreased data and time complexity.
We have implemented our attack on a toy version of `Elisabeth-4` to verify its correctness. It uses the efficient implementation of the Block Wiedemann algorithm of CADO-NFS for the sparse linear algebra.

1 Introduction

Hybrid Homomorphic Encryption (HHE) is an efficient technique for improving the performance of Fully Homomorphic Encryption (FHE) by combining the use of a FHE scheme with the one of a symmetric cipher, e.g., a stream cipher or the combination of a block cipher and an encryption mode of operation. The conducting idea of HHE can be roughly outlined as follows: instead of homomorphically encrypting some data in order to enable their remote processing in an untrusted environment (e.g., a server), a user first encrypts her data using a *classical* symmetric encryption algorithm under a key k and sends a *homomorphic* encryption of k to the remote party. At the cost of an extra computation by the remote party called *transciphering*, this allows to save computation power and bandwidth on the user side as compared with mere FHE, since the ciphertext to plaintext size ratio equals one and at the exception of the homomorphic encryption of k, only symmetric encryption is performed. The transciphering operated

on the remote side transforms the symmetric ciphertext into a homomorphic encryption of the original plaintext under the FHE by running the decryption circuit homomorphically under k.

In this context, `Elisabeth-4` is a HHE-friendly stream cipher introduced by Cosseron et al. at ASIACRYPT 2022 [3]. It is the only fully specified instance of a larger family of HHE-friendly stream ciphers named `Elisabeth`. It is parameterized by a 1024-bit key and an IV of unspecified length. The claimed security level for `Elisabeth-4` is 128 bits. The design of `Elisabeth` extends a few other similar stream ciphers, more specifically `FLIP` [13] and `FiLIP` [12], which have been the subject of a couple of cryptanalytic works, e.g., [5] on `FLIP`.

Stream ciphers in the `Elisabeth` family follow the so-called *Group Filter Permutator* (GFP) paradigm: each keystream symbol belongs to an additive group $\mathbb{G}$ and is derived from a large key by a filtering function that operates on $\mathbb{G}$. Unlike for traditional filtered-LFSR stream ciphers, the filtering function used in `Elisabeth` is differentiated at each clock, most notably in the values of additive constants called *masks*. In the case of `Elisabeth-4`, the group $\mathbb{G}$ equals $(\mathbb{Z}/16\mathbb{Z}, +)$ and we will denote it $(\mathbb{Z}_{16}, +)$ for simplicity.

Two major design goals of the `Elisabeth` ciphers are to optimize the efficiency of the homomorphic decryption and to optimize the efficiency of subsequent format conversions and homomorphic data processings for typical use cases. The distinctive property of `Elisabeth` stream ciphers that they operate in an additive group such as $(\mathbb{Z}_{16}, +)$ rather than extensions of $\mathbb{F}_2$ like in more traditional symmetric ciphers was shown by the designers to play an important role in achieving these designs goals.

Our contribution. In this paper, we present key-recovery attacks on the full `Elisabeth-4`. The starting point for our attack is the observation that over $\mathbb{F}_2$, the least significant bits (LSBs) of all keystream symbols can be expressed as polynomials in the key bits that involve a surprisingly low number of monomials. This observation on the algebraic normal form (ANF) of these keystream bits can be leveraged to mount a simple known-IV linearization attack and reach a first upper bound on the complexity of the key-recovery attack. We then show that various specificities of the `Elisabeth-4` stream cipher allow to substantially reduce the complexity of a more involved linearization-based attack; we list them below.

1. We observe that interactions between the algebraic properties of the two $\mathbb{F}_2$-nonlinear ingredients of the `Elisabeth-4` filtering function (namely, additions over $\mathbb{Z}_{16}$ and *negacyclic* $\mathbb{Z}_{16}$ to $\mathbb{Z}_{16}$ S-boxes) cause *degree and rank defects* in the polynomial equations over $\mathbb{F}_2$ of the LSBs of the keystream symbols. While the attack complexity is only negligibly affected by the defect in the degree of the ANF of the LSB of the keystream symbols (that is shown to be bounded by 12 instead of 16), it is significantly reduced by the highlighted rank defect phenomenon. In order to take into account and leverage the rank defect phenomenon, the linearized equations have to be rewritten in a basis

that consists of polynomials that are not all monomials. This can be viewed as a natural extension of the original linearization technique.
2. We analyze how to leverage the high sparsity of the linearized system (that is not affected by the former rank reduction considerations) in order to reduce the complexity of the key recovery using sparse linear algebra techniques, namely the Block Wiedemann algorithm.
3. We also show that a filtering of the collected equations allows to reduce the size of the linearized system. This decreases the memory complexity of the attack, and the overall time complexity for building and solving the system. In a known IV setting, this is achievable at the cost of an increased data complexity. In a chosen IV setting, this is achieved by a precomputation-based trade-off and allows to decrease the data complexity.

The combination of all these optimizations allows to mount a chosen-IV attack of overall time, data and memory complexity about 2^{88} elementary operations, about 2^{37} nibbles and 2^{54} bits. We list in Table 1 the various attacks described in this paper.

Table 1. Complexities of the key-recovery attacks presented in this paper.

Model	**Time** (operations)	**Data** (nibbles)	**Memory** (bits)	**Reference**
Known IV	2^{124}	2^{43}	2^{87}	Section 3
Known IV	2^{116}	2^{41}	2^{81}	Section 4
Known IV	2^{94}	2^{41}	2^{57}	Section 5
Known IV	2^{88}	2^{87}	2^{54}	Section 6.1
Chosen IV	2^{88}	2^{37}	2^{54}	Section 6.2

In order to corroborate our claims regarding the applicability of rank reduction and Block Wiedemann techniques, etc., we have implemented our attack on a small-scale variant of `Elisabeth-4`. Although this toy cipher is considerably weaker than the original one,[3] it nevertheless preserves most of the structure of the linearized equations of the LSBs of the keystream elements. Our implementation therefore gives some experimental evidence of the validity of our attack. We conducted this practical experiment to reach partial key recovery using the CADO-NFS library [4] and in particular its Block Wiedemann implementation. We have published the code of our implementation on GitHub and provided a Dockerfile to simplify reproducibility:

`https://github.com/jj-anssi/asiacrypt2023-cryptanalysis-elisabeth4`

[3] this toy cipher is not only vulnerable to linearization attacks but also other classes of attacks such as statistical cryptanalysis.

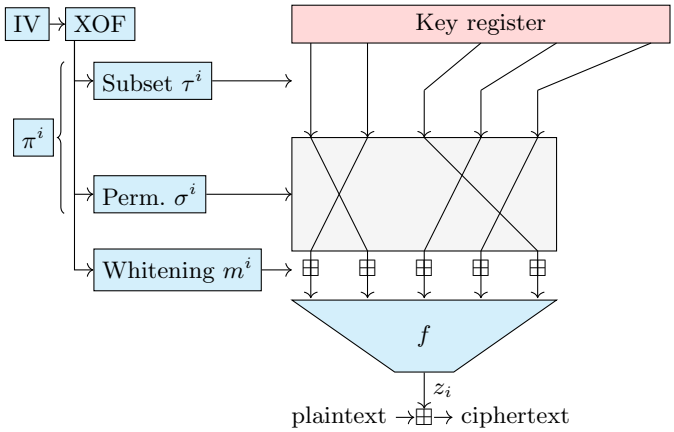

Figure 1. Overall structure of `Elisabeth-4` (figure from [3]). A XOF generates a sequence of parameters from the IV that selects at every step an ordered subset of the secret key register and masks this selection before applying a fixed filtering function to compute the output of `Elisabeth-4` at this step.

Organization. The rest of this paper is organized as follows. We first describe the `Elisabeth-4` stream cipher in Section 2. In Section 3, we present an observation on the number of monomials involved in the ANF of the LSB of all keystream symbols and a simple linearization attack that leverages this observation. In Sections 4, 5 and 6, we present improvements on this basic attack allowed respectively by a rank defect phenomenon, a sparsity of the considered linearized systems and further optimizations of the time and data complexities allowed by restricting the equations considered to build the linearized system. We conclude in Section 7 with the results of our small-scale experiments.

2 Description of `Elisabeth-4`

`Elisabeth-4` is a stream cipher designed by Cosseron et al. and published at ASIACRYPT 2022 [3]. The original specifications introduce a family of stream ciphers, but, for clarity, we only describe the only fully specified instance `Elisabeth-4`, which is the topic of this paper. It is optimized for Hybrid Homomorphic Encryption and the authors claim a security level of 128 bits. The architecture of `Elisabeth-4` (see Figure 1) extends the Improved Filter Permutator principle [13].

Overview. `Elisabeth-4` operates on elements in $\mathbb{Z}/16\mathbb{Z}$, denoted here by $\mathbb{Z}_{16}$. Its state has two components. First, it contains a fixed register loaded with the stream cipher secret key. It is viewed as an array of length $N = 256$ of elements in $\mathbb{Z}/16\mathbb{Z}$, $k = (k_1, \ldots, k_N)$. Secondly, it contains the state of an *extendable output function* (XOF) that is initialized with the stream cipher initialization vector IV.

This state is updated autonomously as `Elisabeth-4` consumes outputs of the XOF as parameters in the generation of its outputs. Consequently, the successive states of the XOF and all of its outputs can be considered as public values.

At Step i, `Elisabeth-4` generates an element in $\mathbb{Z}_{16}$ in the following manner. Using the XOF, an ordered *arrangement* of length $r \cdot t = 60$, *i.e.* a $r \cdot t$-tuple of distinct elements of $\{1, \ldots, N\}$, is selected. We denote it by $\pi^i = \left(\pi^i_1, \ldots, \pi^i_{r \cdot t}\right)$. This arrangement can be seen as the composition of the selection of a $r \cdot t$-subset τ^i of $\{1, \ldots, N\}$, and a permutation σ^i of its elements. The XOF also produces a whitening vector $m^i = \left(m^i_1, \ldots, m^i_{r \cdot t}\right)$ in $\mathbb{Z}_{16}^{r \cdot t}$. The keystream element $z_i \in \mathbb{Z}_{16}$ at the output of Step i is obtained by applying a fixed filtering function f to the sum of the key elements selected by the arrangement and the whitening elements:

$$z_i = f(k_{\pi^i_1} + m^i_1, \ldots, k_{\pi^i_{r \cdot t}} + m^i_{r \cdot t}).$$

The filtering function f. The filtering function f internally uses $t = 12$ parallel calls to a function g applied on $r = 5$ elements. The r outputs are elements of $\mathbb{Z}_{16}$ and are summed together to produce the output of f (see Figure 2). The function g itself uses a nonlinear function h that we describe below.

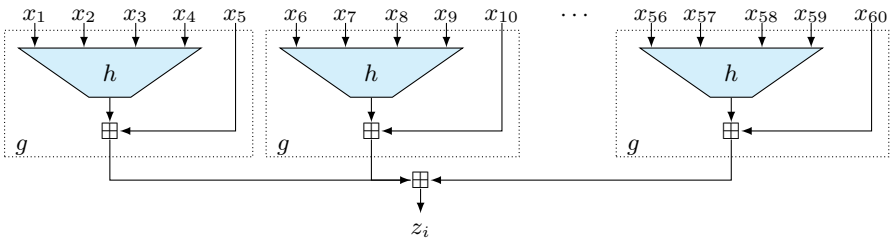

Figure 2. The function f in `Elisabeth-4` internally uses $t = 12$ calls to g, which is in turn defined using the nonlinear function h. Every element belongs to $\mathbb{Z}_{16}$.

The function g. The 5-to-1 function g is constructed as the sum of a nonlinear 4-to-1 function h and a linear function of the remaining variable (see Figure 2):

$$\begin{array}{rccc} g & : & \mathbb{Z}_{16}^5 & \longrightarrow & \mathbb{Z}_{16} \\ & & (x_1, \ldots, x_5) & \longmapsto & h(x_1, x_2, x_3, x_4) + x_5\,. \end{array}$$

The construction of the filtering function f over h is depicted in Figure 2. The function h uses eight *negacyclic* look-up tables $S_1, \ldots, S_8$ over $\mathbb{Z}_{16}$ (see Figure 3). These NLUTs were selected at random by the designers [3]. They are given in Appendix A.

Definition 1. *A negacyclic lookup table (NLUT) [1] over $\mathbb{Z}/2^\ell\mathbb{Z}$ is a lookup table S of length 2^ℓ that verifies the following property:*

$$\forall i \in [0, 2^{\ell-1} - 1], \quad S[i + 2^{\ell-1}] = -S[i].$$

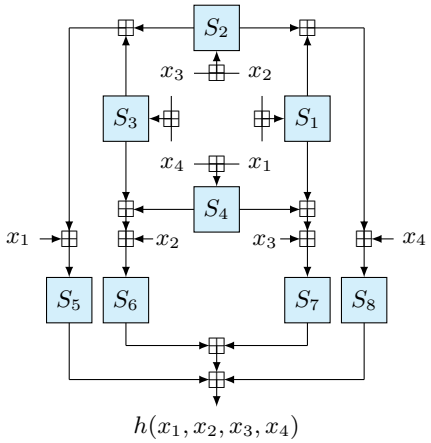

Figure 3. The function h uses eight NLUTs S_i to map the four variables (x_1, x_2, x_3, x_4) to $h(x_1, x_2, x_3, x_4)$.

3 Basic attack

In this section, we present a general overview of the linearization principle before describing a basic application to the cryptanalysis of `Elisabeth-4`.

3.1 Linearization technique

Algebraic cryptanalysis is a classical cryptanalysis paradigm. It consists in writing multivariate polynomial equations linking the secrets of a cryptographic mechanism (e.g., its key) and what is known to the attacker (e.g., IV bits, keystream bits, plaintext, ciphertext, etc.), and then solving this system of algebraic equations to recover the secrets. In principle, this can always be achieved by putting into equations the relations stemming from the specification of the cryptographic mechanism. The resulting equations can be difficult to write and/or store in practice, or require to introduce intermediate variables, but in the case where it is possible, the complexity of solving a large nonlinear multivariate system of equations in a large number of variables is often impractical. Let us consider the following formalization. Let us denote $\mathbb{Z}/2\mathbb{Z}$ by $\mathbb{Z}_2$. We denote the n-bit secret the cryptanalyst aims at deriving by $(x_1, \ldots, x_n) \in \mathbb{F}_2^n$. For any element $u = (u_1, \ldots, u_n) \in \mathbb{Z}_2^n$, we denote by x^u the monomial $x_1^{u_1} \cdots x_n^{u_n}$. Due to the field equation $x^2 = x$ over $\mathbb{F}_2$, any Boolean function p in n variables can be written uniquely as a sum of such monomials:

$$p(x_1, \ldots, x_n) = \sum_{u \in \mathbb{Z}_2^n} a_u x^u, \quad a_u \in \mathbb{F}_2 \,.$$

This representation is the *algebraic normal form* (ANF) of p. In the case of a stream cipher generating a binary keystream, every keystream bit provides a

polynomial equation in the key bits (or, in another typical setting not considered in the sequel, in the initial state value). An attacker can therefore compute the ANFs of the Boolean functions that take as input the key bits and return given keystream bits, collect these keystream bits, and construct an overdetermined system of nonlinear equations. If the attacker is able to solve this system, she recovers the key.

Although solving a system of polynomial equations is hard in general, several methods exist, e.g. solving methods based on Gröbner basis computation technique such as Faugères' F4 and F5 algorithms [6,7]. An elementary method of resolution is linearization [8]. Considering every monomial appearing in the system as an independent variable, the system can be viewed as a linear system of equations involving a larger number of variables. In order for the linear system to be solvable, a large number of equations needs to be available. The ANF of a Boolean function in n variables can count up to 2^n monomials, so this technique cannot be applied with profit in the general case. However, in the case of a stream cipher whose design is based on the filtering of a linear register, two factors make it worth pursuing this type of attacks. First, the degree of the ANF of the equations relating key bits and keystream bits does not grow and stays bounded by the degree of the filtering function. This enforces an upper bound μ on the number of monomials involved in the system, and thus on the number of variables in the linearized system. This number of monomials can be further restricted by the structure of the polynomial function. Secondly, the number of available equations is not limited since the size of the keystream can in general be adapted to collect enough equations, without requiring the introduction of additional variables, to reach a full-rank linear system with good probability. This system can then be solved in about μ^ω operations, where $2 \leq \omega \leq 3$ is the linear algebra exponent representing the complexity of matrix multiplication. In particular, for μ such that $\mu^\omega < 2^\kappa$ with κ the security parameter in bits, an attack is found.

Suppose the attacker has collected δ keystream bits z_i, $1 \leq i \leq \delta$, and computed the corresponding δ ANFs which we denote by $p_i(x_1, \ldots, x_n) = \sum_{u \in \mathbb{Z}_2^n} a_u^i x^u$ involving at most μ monomials. We let $\mathbf{z} \in \mathbb{F}_2^\delta$ denote the vector of keystream bits and $\mathbf{A}$ to be the $\mathbb{F}_2$-matrix of size $\delta \times \mu$ constructed as $\mathbf{A}[i, u] = a_u^i$. Finding the solutions to the matrix equation $\mathbf{Ax} = \mathbf{z}$ enables to recover monomials in the key bits, and the key itself since the key bits are in general part of the monomial basis.

3.2 Basic linearization attack on `Elisabeth-4`

Our attack on `Elisabeth-4` relies on solving a system of Boolean polynomial equations on the key bits using the linearization method described in Section 3.1.

Without loss of generality, we consider a keystream generated from a single known IV. At iteration i of the keystream generator, a subset τ^i of the key register, a permutation σ^i of this subset and a whitening vector m^i are derived from the IV. Then, the filtering function f is applied. We can thus consider the ANF of each Boolean function that takes as input the vector of $r \cdot t = 60$ key nibbles and returns one of the bits of the keystream element. In particular, in our

attack, we consider the Boolean function f_i that returns the least significant bit (LSB) of the keystream element. We indeed observe that the LSB of the addition in $\mathbb{Z}_{16}$ behaves linearly in $\mathbb{F}_2$. Thus, since f is constructed as the modular sum of $t = 12$ applications of the function g, where g is constructed as the sum of h, a nonlinear function of $r - 1 = 4$ elements, and a fifth element, the ANF of each f_i can be written as the sum of

- the least significant bits of the 12 key nibbles selected for the final addition in the g function (see Figure 2),
- the algebraic normal form of 12 Boolean *variations* of the function h, restricted to the LSB of its output, and parameterized by the whitening mask values applied on the selected ordered 4-tuple of key elements.

If we fix any unordered quartet $\{x_1, x_2, x_3, x_4\}$ of four distinct key elements, the possible variations of h that take the elements of this set as inputs can be described as the set of Boolean functions $\{h_{m,\sigma}\}_{m\in\mathbb{Z}_{16}^4,\sigma\in\mathfrak{S}_4}$, with $m = (m_1, m_2, m_2, m_3) \in \mathbb{Z}_{16}^4$, $\mathfrak{S}_4$ the permutation group over four elements, and

$$\begin{aligned} h_{m,\sigma} : \quad \mathbb{Z}_{16}^4 \quad &\longrightarrow \quad \mathbb{F}_2 \\ (x_1, \ldots, x_4) &\longmapsto \mathrm{LSB}(h(x_{\sigma(1)} + m_1, x_{\sigma(2)} + m_2, x_{\sigma(3)} + m_3, x_{\sigma(4)} + m_4)) \,. \end{aligned}$$

In the following, note that while `Elisabeth-4` relies on operations defined on $\mathbb{Z}_{16}$, we write the algebraic modelization in $\mathbb{F}_2$, using the LSB of each keystream element.

Bounding the number of monomials. Since each element in $\mathbb{Z}_{16}$ is written on four bits, the observation above shows that the degree of the ANF of each f_i is bounded by $4 \cdot 4 = 16$. Moreover, the number of monomials required to describe the ANF of all LSB of the outputs of `Elisabeth-4` is further reduced by the element-wise structure of the cipher. Indeed, no monomial can involve key variables appearing in more than four key elements, since the only $\mathbb{F}_2$-nonlinear functions involved are h variations that depend on four key elements. The four variables are written on a total of 16 bits. Thus, at most 2^{16} monomials can be formed with the bits of every selection of an unordered quartet of key elements regardless of the h variation $h_{m,\sigma}$ applied to this unordered quartet. This implies that the number of monomials to consider can be upper bounded by $\binom{256}{4} \cdot 2^{16} = 2^{43.4}$ monomials.[4]

Construction of the linearized system. We describe here the construction of the matrix $\mathbf{A}$. We begin by the precomputation of the ANFs of the $2^{16} \cdot 4!$ variations $h_{m,\sigma}$ of h by computing their truth table and applying the fast Möbius

[4] This direct bound can be further refined by taking into account that monomials involving less than four elements are counted multiple times. Observing that the number of monomials involving variables from exactly j elements is 15^j, a finer bound on the number of monomials is $\sum_{j=0}^{4} \binom{256}{j} 15^j \approx 2^{43.0}$.

transform. This step has a complexity of around $4! \cdot 2^{32} \approx 2^{36.6}$ applications of the h function to compute the truth table, $4! \cdot 2^{16} \cdot \log(2^{16}) \cdot 2^{16} \approx 2^{40.6}$ elementary operations to compute the fast Möbius transform, and storing the result requires $2^{36.6}$ bits of memory. To build the matrix $\mathbf{A}$ of the linearized system, we associate to every unordered quartet of key elements a dedicated set of 2^{16} monomials.[5] Thus, the matrix $\mathbf{A}$ has $\mu = \binom{256}{4} \cdot 2^{16}$ columns. The matrix $\mathbf{A}$ is then built row by row, by considering successively least significant bits of the keystream. For a given row and for each of the $t = 12$ applications of the g function:

- we determine from the XOF outputs the unordered quartet τ of key elements on which a variation of function h is applied, and its parameters m and σ. The columns of monomials associated to the set of elements τ are set in the row of the matrix A according to the precomputed ANF of $h_{m,\sigma}$.
- we determine the key element that is linearly added to the output of h (see Figure 2) and set a column associated with the LSB of this element in the set of monomials associated to a quartet containing this key element.

Finally, the sum of the LSB values of the whitening masks added to the fifth inputs of the g function calls determine whether an additional column corresponding to a constant monomial should be set in the row. Thus, at most $t \cdot (2^{16} + 1) + 1$ bits are non-zero in every row. Building the matrix $\mathbf{A}$ requires around $\delta \cdot t \cdot 2^{16}$ elementary operations.

Description of the attack. We set $\delta \gtrapprox \mu$. The data complexity of the attack is thus $\delta \approx 2^{43.3}$ nibbles and the time complexity of building the matrix is around $2^{62.9}$ elementary operations. The system is then solved by standard linear algebra techniques for a cost μ^ω. With straightforward Gaussian elimination ($\omega = 3$), the time complexity of the resolution is 2^{129} elementary operations, and its memory complexity, which corresponds to the cost of storing $\mathbf{A}$, is about $\delta \cdot \mu = 2^{87}$ bits. Using Strassen algorithm where $\omega = \log_2(7)$ and a small multiplicative constant < 6 needs to be taken into account [14], the time complexity can be brought down to less than $2^{124.14}$ elementary operations. We make the (customary in linearization attacks) assumption that the extra cost of recovering the key bits after recovering the affine space of solutions is negligible.[6]

This basic attack is a known-IV attack. It can be modified into a chosen-IV attack with a significantly improved online time complexity but similar total time and memory complexity. Indeed, one can simply precompute the matrix $\mathbf{A}$ and its Gaussian elimination (or LU decomposition in the case of Strassen algorithm) before gathering the data and computing the solutions via a matrix-vector multiplication.

[5] In the finer monomial representation, without monomial duplication, the association between the parameters of the h function and the set of monomials is more complex, but can still be achieved with some indexes bookkeeping.

[6] We do not not enter here into a discussion on the dimension of the found affine space of solutions and how a large number of key bits can be derived from any particular solution and leveraged to efficiently recover the missing key bits since this would largely amount to anticipating the analysis of the next section.

4 Improving the attack using rank defects

In this section, we identify an unexpected rank defect of the $2^{16} \cdot 4! \times 2^{16}$ matrix whose rows are the ANFs of the functions $h_{m,\sigma}$. We then discuss the impact of this rank defect on the linearization attack described in Section 3, namely a lowering of its expected complexity. Finally, we partially explain this rank defect with theoretical arguments, relating it to the use of negacyclic Sboxes.

4.1 Identification of a rank defect

As mentioned in Section 3, the ANFs of the $2^{16} \cdot 4!$ variations of h can be precomputed and stored in a matrix denoted here by $\mathbf{H}$ at a practical cost. Since the rank of $\mathbf{A}$ is related to the dimension of the vector space spanned by these ANFs seen as vectors of coefficients, we have computed the rank of the matrix $\mathbf{H}$ and obtained $\rho = 8705 \approx 2^{13.1}$, which is lower that the expected 2^{16}. Performing a reduced row echelon form computation, we can write

$$\mathbf{LH} = \begin{bmatrix} \mathbf{P} \\ 0 \end{bmatrix}, \quad \mathbf{H} = \mathbf{L}'\mathbf{P},$$

with $\mathbf{P}$ an upper triangular matrix of size $\rho \times 2^{16}$, $\mathbf{L}$ a square invertible matrix of size $2^{16} \cdot 4!$ that is the product of elementary matrices and $\mathbf{L}'$ is the $2^{16} \cdot 4! \times \rho$ matrix obtained as the first ρ columns of the inverse of the matrix $\mathbf{L}$. $\mathbf{P}$ can be seen as a change of 'basis' matrix: its rows define ρ linear combinations of the monomials which are sufficient to describe the rows of matrix $\mathbf{H}$. We will refer to these combinations as the *polynomial basis*.

Given the greater value of small degree monomials, we adopt a degree monomial order and reorder the columns of $\mathbf{H}$ accordingly before performing the reduced row echelon reduction. With this choice, we observe that the polynomial basis contains 12 monomials corresponding to input bits to the h function. The most significant bits (MSBs) of the 4 input elements of h however do not belong to the span of the polynomial basis.

4.2 Impact on the linearization attack

Since the variations $h_{m,\sigma}$ of the h functions only linearly depend on the ρ polynomials of the polynomial basis, we adapt the linearization attack in a straightforward way. Instead of associating to every unordered quartets a set of 2^{16} monomial variables, we associate a set of ρ polynomial variables. Thus, the number of columns of the matrix $\mathbf{A}$ decreases to $\mu = \binom{256}{4} \cdot \rho \approx 2^{40.5}$, and the matrix can be built in the same way as before by using the rows of the matrix $\mathbf{L}'$ instead of those of the matrix $\mathbf{H}$. Linear terms corresponding to the LSBs of the fifth input to the function g can be handled as before since they are part of the polynomial basis. The number of equations required for the system to be solvable is reduced accordingly to $\delta \gtrapprox \mu$ using heuristics that the dimension of the affine space of solutions is thereby reduced to a sufficiently small number

of vectors. This is discussed in Appendix D and backed-up by the experimental results described in Section 7.

Thus, the rank defect lowers the complexity of solving the linear system and consequently the overall time complexity of the attack. The computation of the row echelon form of $\mathbf{H}$ and the subsequent computation of $\mathbf{L}'$ require about $\rho \cdot 2^{16} \cdot 4! \cdot 2^{16} \approx 2^{48.43}$ operations. The data complexity of the attack is $\mu \approx 2^{40.5}$, its time complexity is μ^ω, which is about $2^{121.4}$ elementary operations using Gaussian elimination and $2^{116.2}$ elementary operations using Strassen algorithm, and its memory complexity is $\mu^2 \approx 2^{80.9}$.

The resolution of the system enables to recover for all the unordered quartets of key elements the values taken by the polynomials of the polynomial basis applied to the relevant key bits. As seen above, this directly provides the key bits, except for the most significant bits of the key elements. These can be recovered quartet by quartet by small 2^4 exhaustive searches considering the system of ρ algebraic equations provided by the polynomial basis. In total, this adds $\binom{256}{4}2^4$ elementary operations to our time complexity.

4.3 Explaining the rank defect

In this section, we provide a partial explanation for the rank defect of the matrix $\mathbf{H}$ whose rows correspond to the ANFs of the $2^{16} \cdot 4!$ variations $\{h_{m,\sigma}\}_{m \in \mathbb{Z}_{16}^4, \sigma \in \mathfrak{S}_4}$ of h. We find that this phenomenon is largely due to the interaction between modular addition and negacyclic look-up tables. Whilst analyzing said interaction, we also proved that the degree of the LSB at the output of the function h is bounded by 12 rather than 16, and thus that the degree of the LSB of the keystream is also bounded by 12. Although this result did not lead to a significant improvement of our linearization attack,[7] we provide its proof in Annex B.[8] Note that this proof uses definitions and propositions introduced in the rest of this section.

On negacyclic look-up tables. We start with two observations on negacyclic look-up tables over $\mathbb{Z}_{2^\ell}$. In `Elisabeth-4`, negacyclic look-up tables are applied to the sum of a key word and a whitening word. The following proposition shows that the composition of an addition with a negacyclic look-up table (NLUT) does not change its negacyclic nature: the composition is still a NLUT.

Proposition 1. *Let ℓ be an integer and let S be a NLUT over $\mathbb{Z}_{2^\ell}$. For any $\tilde{x} \in \mathbb{Z}_{2^\ell}$, the look-up table S' defined as $S'[x] \coloneqq S[x + \tilde{x}]$ for all $x \in \mathbb{Z}_{2^\ell}$ is negacyclic.*

[7] Indeed, the number of monomials in 16 variables of degree at most 12 is equal to $\sum_{i=0}^{12} \binom{256}{i} = 2^{15.98}$ which is not significantly smaller than 2^{16}.

[8] Note that an observation made thanks to experiments on the degree of the keystream LSB for some mask values had already been made by the authors of `Elisabeth-4` [3], but here we provide a proof that holds for any mask.

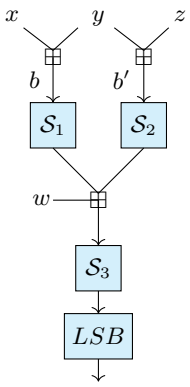

Figure 4. The Antler function $H_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}$.

Proof. For all $x \in \mathbb{Z}_{2^\ell}$, $S'[x+2^{\ell-1}] = S[x+\tilde{x}+2^{\ell-1}] = -S[x+\tilde{x}] = -S'[x]$. □

The next proposition highlights a property of NLUTs that we will show to be largely responsible for the rank (and degree) defect(s).

Proposition 2. *Let ℓ be an integer and let S be a negacyclic look-up table over $\mathbb{Z}_{2^\ell}$. The Boolean function $x \mapsto LSB(S[x])$ does not depend on the MSB of x.*

Proof. For any $y \in \mathbb{Z}_{2^\ell}$, y and $-y$ have the same parity and thus the same LSB. As a consequence, for any $x \in \mathbb{Z}_{2^\ell}$, $S[x]$ and $S[x+2^{\ell-1}] = -S[x]$ have the same LSB. Since x and $x+2^{\ell-1}$ have all their bits equal except for their MSB, the proposition holds. □

On Antler functions. In the sequel, we rely on the analysis of a family of 16-bit to 1-bit functions with a 2-round, triangular structure that we propose to name *Antler functions* (see Figure 4). This is because the LSB of the function h and, as we will see in the sequel, the LSB of any of its variations $h_{m,\sigma}$, can be easily shown to be the sum of four Antler functions. Antler functions can be defined as follows.

Definition 2. *Let $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$ be 3 negacyclic look-up tables. The Antler function associated to $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$ is defined as*

$$\begin{array}{rccc} H_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3} & : & \mathbb{Z}_{16}^4 & \longrightarrow & \mathbb{F}_2 \\ & & (x,y,z,w) & \longmapsto & LSB(\mathcal{S}_3\left[\mathcal{S}_1[x+y]+\mathcal{S}_2[y+z]+w\right]). \end{array}$$

Although Antler functions are defined with domain $\mathbb{Z}_{16}^4$, we view them as Boolean functions by considering the binary representation of elements in $\mathbb{Z}_{16}$ and thus assimilating $\mathbb{Z}_{16}^4$ to 16-bit words or equivalently elements in $\mathbb{F}_2^{16}$. Generally, in the following, we assimilate elements in $\mathbb{Z}_{16}^n$ to $4n$-bit words or equivalently elements in $\mathbb{F}_2^{4n}$. Antler functions possess properties inherited from NLUTs. In particular, a direct consequence of Proposition 1 is the following proposition.

Proposition 3. *Let $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$ be 3 negacyclic look-up tables and $(\tilde{x}, \tilde{y}, \tilde{z}, \tilde{w}) \in \mathbb{Z}_{16}^4$. Then, $H_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}(x+\tilde{x}, y+\tilde{y}, z+\tilde{z}, w+\tilde{w})$ is also an Antler function.*

Proposition 3 shows that the property that the LSB of h can be expressed as a sum of four Antler functions extends to all variations in the set $\{h_{m,id}\}_{m\in\mathbb{Z}_{16}^4}$, where $id \in \mathfrak{S}_4$ is the identity permutation. In particular, the vector space spanned by the set of ANFs of $\{h_{m,id}\}_{m\in\mathbb{Z}_{16}^4}$ is a subspace of the vector space spanned by the ANFs of all Antler functions. This can be generalized as follows. The vector space spanned by the set of ANFs of $\{h_{m,\sigma}\}_{m\in\mathbb{Z}_{16}^4,\sigma\in\Sigma_4}$ is a subspace of the vector space spanned by the ANFs of all Antler functions for any ordering of their input variables. Thus, to bound the dimension of the vector space spanned by $\{h_{m,\sigma}\}_{m\in\mathbb{Z}_{16}^4,\sigma\in\mathfrak{S}_4}$, we first exhibit a bound on the dimension of the vector space spanned by all Antler functions.

In the sequel, we will exhibit upper bounds on the dimension of some vector spaces of Boolean functions. We introduce the following definition.

Definition 3. *The rank of a set $\{H_i\}_{i\in I}$ of Boolean functions in n binary variables is defined as the dimension of the vector space of functions spanned by this set.*

For example, the rank of the set of variations $h_{m,\sigma}$ of h is the rank of $\mathbf{H}$.

Bounding the rank of the set of Antler functions.

Proposition 4. *The rank of the set of Antler functions is bounded by $2^{10.43}$.*

To prove this proposition, we rely on the following lemma.

Lemma 1. *Let $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$ be 3 negacyclic look-up tables. Let $G_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}$ be the Boolean function defined as*

$$\begin{aligned} G_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3} \quad : \quad \mathbb{Z}_{16}^3 &\longrightarrow \mathbb{F}_2 \\ (b, b', w) &\longmapsto LSB(\mathcal{S}_3\left[\mathcal{S}_1[b] + \mathcal{S}_2[b'] + w\right]). \end{aligned}$$

Then, the rank of the set of Antler functions is bounded by the dimension of the vector space spanned by all functions $G_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}$.

Sketch of proof. Let $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$ be 3 negacyclic look-up tables. It is easy to see that $H_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3} = G_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3} \circ F$ where F is defined as

$$\begin{aligned} F \quad : \quad \mathbb{Z}_{16}^4 &\longrightarrow \mathbb{Z}_{16}^3 \\ (x, y, z, w) &\longmapsto (b = x + y, b' = y + z, w = w). \end{aligned}$$

Thus, consider a basis of the set of all functions of the form $G_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}$. It is easy to see that the set of all functions constructed as the composition of F with a function in this basis generates the vector space spanned by Antler functions. Lemma 1 is thus shown. □

To prove Proposition 4, we thus only need showing that the rank of the set of all functions $G_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}$ is bounded by $2^{10.43}$. In this section, we only demonstrate a simpler result, namely, a bound 2^{11} on the dimension, since it gives a first intuition of the causes of the rank defects but remains simple to explain. We refer to Annex C for a full proof for the bound $2^{10.43}$ of Proposition 4. We believe the actual rank to be even lower, namely $1088 = 2^{10.09}$, as suggested by our experiments. We did not manage to explain theoretically this tighter bound.

Proof of the bound 2^{11}. Notice that $G_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}$ has domain $\mathbb{Z}_{16}^3$. However, by Proposition 2, the output of $G_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}$ does not depend on the most significant bit of w. Thus, it can be seen as a Boolean function that takes as input 11 bits. This trivially implies that the dimension of the space spanned by all functions $G_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}$ is at most 2^{11}. □

Bounding the rank of H. We have shown that the rank of the set of Antler functions for a fixed ordering of the input variables is bounded by $2^{10.43}$. Since these functions take as input four elements in $\mathbb{Z}_{16}$, the rank of this set for any ordering is at most $4! \cdot 2^{10.43}$. However, the vector space spanned by the set containing the functions $H_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}$ for all NLUTs $(\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ is equal to the vector space spanned by the compositions $(x, y, z, w) \mapsto H_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}(z, y, x, w)$ of $H_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}$ with a transposition of x and z. Thus, the vector space spanned by all Antler functions and for any ordering of their variables has dimension at most $\frac{4!}{2} 2^{10.43} \approx 2^{14.01}$. Thus, the rank of the set $\{h_{m,id}\}_{m \in \mathbb{Z}_{16}^4, \sigma \in \mathfrak{S}_4}$ is at most $2^{14.01}$.

As mentioned above, using experiments, we found that the rank of an Antler function is in fact bounded by 1088. This gives a bound of $\frac{4!}{2} \cdot 1088 \approx 2^{13.67}$ for the rank of $\{H_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3} \circ \sigma\}$ for all NLUTs and all permutations, which can be considered close to the bound $8705 \approx 2^{13.1}$ as this bound holds for the particular case of `Elisabeth-4` (in which NLUTs are repeated across applications of Antler functions within h).

5 Solving sparse linear systems

A matrix is said to be *sparse* if the number of its nonzero coefficients is small with regards to the total number of coefficients. The matrix $\mathbf{A}$ of size $\delta \times \binom{256}{4} \cdot \rho$ obtained in our attack is sparse since it has at most $t(\rho + 1) \ll \binom{256}{4} \cdot \rho$ nonzero coefficients per $\binom{256}{4} \cdot \rho$-bit row. Binary sparse matrices can be stored in a compressed form, e.g., by storing for every row the indexes of the columns containing a nonzero coefficient. Further, sparse linear algebra algorithms can take advantage of this property of the matrix to solve classic linear algebra problems at a lower complexity.

In this section, we describe methods that allow to efficiently solve sparse linear algebra problems before detailing an improved linearization attack.

5.1 Efficient methods for sparse linear algebra

As mentioned above, sparse linear algebra problems can be solved more efficiently than general linear algebra problems. A well-known efficient method to solve sparse linear problems is the 1986 Wiedemann algorithm [15]. This algorithm, which exploits the cheapness of sparse matrix-vector products, has been widely studied [2, 9, 10]. In our attack, we use a variant of Wiedemann algorithm, the Block Wiedemann algorithm (BW) [2]. This algorithm introduced by Coppersmith in 1994 converts Wiedemann algorithm to a block form, allowing to process several bits at the same time. In this section, we begin with a brief description of the Wiedemann and BW algorithms before computing the improved complexity of our attack.

In the following, we let $\mathbf{M}$ be a square $\mathbb{F}_2$-matrix of dimension n and $\mathbf{y}$ be a vector in $\mathbb{F}_2^n$. We assume that $\mathbf{M}$ is sparse, *i.e.* that the number of nonzero coefficients per row is bounded by $s \ll n$. We wish to find a solution to the equation

$$\mathbf{M}\mathbf{x} = \mathbf{y}. \tag{1}$$

Wiedemann algorithm. We begin with a brief overview of the Wiedemann algorithm [15]. We refer to specialized papers for a more comprehensive description of this algorithm, e.g., [2, 9, 10, 15]. Wiedemann algorithm must be fed a square matrix $\mathbf{M}$ of size $n \times n$ and a vector $\mathbf{y}$ and returns a solution $\mathbf{x}^0$ or the error symbol. It requires at most $3n$ multiplications of the matrix $\mathbf{M}$ with a vector, $\mathcal{O}(n^2 \log(n))$ other operations, and a limited amount of storage in addition to the cost of storing $\mathbf{M}$ (see [10]).

This algorithm is divided into two main steps. In a first step, the algorithm recovers the minimal polynomial $f_{\mathbf{M}}$ of the matrix $\mathbf{M}$. Then, it uses this minimal polynomial to recover a solution to (1). Indeed, recovering $f_{\mathbf{M}}$ provides non-trivial coefficients $(c_i)_{0 \le i \le n}$ such that

$$\sum_{i=0}^{n} c_i \mathbf{M}^i = 0. \tag{2}$$

If $\mathbf{M}$ is nonsingular, then necessarily $c_0 \neq 0$. The unique solution can then be expressed as $\mathbf{x} = -(1/c_0) \sum_{i=1}^{n} c_i \mathbf{M}^{i-1}\mathbf{y}$. On the other hand, if $\mathbf{M}$ is known to be singular and $\mathbf{y} = \mathbf{0}$, the algorithm finds a vector in the kernel of $\mathbf{M}$ in the following manner. Since $\mathbf{M}$ is singular, $c_0 = 0$. Letting c_δ be the first nonzero coefficient and $\mathbf{r}$ be a random vector, it comes that computing the successive vectors $\mathbf{M}^i \left(\sum_{i=0}^{n} c_i \mathbf{M}^{i-\delta}\mathbf{r}\right)$ for $i = 1, \ldots, \delta$ yields a kernel element as long as $\mathbf{r}$ is not in the kernel of the non-null matrix $\sum_{i=0}^{n} c_i \mathbf{M}^{i-\delta}\mathbf{r}$, which happens with probability greater than $1/2$. Since $\mathbf{M}$ is s-sparse, a matrix-vector multiplication costs at most $2sn$ operations. The total complexity of this step can thus be bounded by $n(2sn + 2n) = 2(s+1)n^2$ operations.

To recover the minimal polynomial $f_{\mathbf{M}}$ of the matrix $\mathbf{M}$, the algorithm uses the sequence $(\mathbf{u}^{\mathrm{T}}\mathbf{M}^i\mathbf{v})_{i \ge 0}$ for some random $\mathbf{u}$ and $\mathbf{v}$.[9] Since $\sum_{i=0}^{n} c_i \mathbf{M}^i \mathbf{v} = \mathbf{0}$, *a*

[9] In practice, if $\mathbf{M}$ is known to be nonsingular, one uses $\mathbf{v} = \mathbf{y}$.

fortiori, for any vectors $\mathbf{u}, \mathbf{v}$ and any integer j, $\sum_{i=0}^{n} c_i \mathbf{u}^{\mathrm{T}} \mathbf{M}^{i+j} \mathbf{v} = 0$. Therefore, the sequence $(\mathbf{u}^{\mathrm{T}} \mathbf{M}^i \mathbf{v})_{i \geq 0}$ is a linear recurrent sequence of degree smaller or equal to n. Thus, one can compute its first $2n$ first terms, feed them to the Berlekamp-Massey algorithm [11] and obtain the minimal polynomial f_1 of this sequence. Since $f_{\mathbf{M}}$ cancels this sequence, it comes that $f_1 | f_{\mathbf{M}}$. In fact, one can show that with high probability, $f_1 = f_{\mathbf{M}}$ After a few tries using different random vectors, the algorithm recovers $f_{\mathbf{M}}$ and thus provides a solution to (1).[10]

Block-Wiedemann algorithm. Coppersmith's Block-Wiedemann algorithm [2] is a probabilistic algorithm that allows to parallelize or distribute the Wiedemann algorithm. It takes as input a square matrix $\mathbf{M}$ and returns a solution of the equation $\mathbf{Mx} = \mathbf{0}$ or the error symbol. In this algorithm, one draws simultaneously l_1 random vectors for $\mathbf{u}$ and l_2 random vectors for $\mathbf{v}$ and considers the sequence $(\mathbf{M}_i)_{i \geq 0}$ such that

$$\mathbf{M}_i = \mathbf{U}^{\mathrm{T}} \mathbf{M}^i \mathbf{V} \in \mathbf{F}_2^{l_1 \times l_2} \text{ where } \mathbf{U}^{\mathrm{T}} \in \mathbf{F}_2^{l_1 \times n} \text{ and } \mathbf{V} \in \mathbf{F}_2^{n \times l_2}.$$

In practice, we will set $l_1 = l_2$ and this parameter is usually equal to the size of a word on the processor considered. In the binary case, the implementation can also take advantage of the degree of parallelism offered by the execution of binary instructions on the bits of a word by a processor. Thus we typically use $l_1 = 64$. Coppersmith proposes a generalization of the Berlekamp-Massey algorithm that allows to compute a linear recurrence that generates this sequence. He then notices that the sequence of $(\mathbf{M}_i)_{i \geq 0}$ is in fact determined by its first $n/l_1 + n/l_2 + \mathcal{O}(1)$ elements rather than $2n$. This algorithm allows to compute a solution to an homogeneous sparse linear system in time $3n/l_1$ matrix-vector products, which cost at most $2sn$, which must be added to a small term in $\mathcal{O}(n^2)$ operations [2]. The total complexity can thus be approximated by $6sn^2/l_1$ operations.

Solving arbitrary sparse linear systems. It is always possible to transform the problem of solving the equation $\mathbf{Mx} = \mathbf{y}$ for arbitrary $\mathbf{M}$ (in particular $\mathbf{M}$ non-square) and $\mathbf{y}$, into a problem that can be fed to either algorithms. If $\mathbf{M}$ is not square, several methods (some heuristic, some not) allow to transform the problem into a problem where the matrix considered is square, and in a way such that a solution to the square problem will provide a solution to the original problem with high probability [9]. We will not go into details about how these methods work. Further, to transform a non-homogeneous system into a homogeneous one, one can simply consider the equivalent problem of finding the solutions $\mathbf{x}^0$ to the equation

$$(\mathbf{M}|\mathbf{y})\mathbf{x} = \mathbf{0}$$

such that $\mathbf{x}_n^0 \neq 0$. If the matrix $(\mathbf{M}|\mathbf{y})$ is not square, one must then transform it into a square matrix using the methods mentioned above.

[10] Although the algorithm is in practice slightly more complicated, we do not go into details as this is not the core of the article.

5.2 Improved attack

The matrix $\mathbf{A}$ is a sparse matrix of size $\delta \times \mu$, where $\mu = \binom{N}{4} \cdot \rho$, $\delta \gtrapprox \mu$ and the number of nonzero coefficients per row is bounded by $s = t \cdot (\rho+1)+1 \approx t \cdot \rho$. In our attack, we wish to recover the kernel of the matrix $\mathbf{A}|\mathbf{z}$. After extending $\mathbf{A}|\mathbf{z}$ to a square matrix of size about δ for a negligible cost, recovering vectors in its kernel has complexity $6 \cdot s \cdot \delta^2 / l_1$ elementary operations. With $l_1 = 64$, we improve the time complexity of the attack described in Section 4.2 from $2^{116.2}$ elementary operations using Strassen algorithm to $2^{94.2}$ using the Block Wiedemann algorithm. The memory complexity is also improved from $2^{80.9}$ to $s \cdot \delta \approx 2^{57.14}$, as the matrix can be stored more efficiently using its sparsity. The data complexity is not affected.

6 Filtering the equations used by the linearization attack

In this section, we propose a final improvement to our initial attack by filtering the collected equations in order to reduce the number of variables in the considered linearized systems. This leads to a known-IV time-data tradeoff, that reduces the time complexity of the attack at the cost of an increased data complexity, and a chosen-IV attack with a decreased time and data complexity thanks to an offline precomputation selecting appropriate IVs.

Contexts where a chosen-IV attack is possible usually correspond to a scenario where the attacker has access to a chosen ciphertext decryption/verification oracle. In the HHE context, a client must not only be able to encrypt some data using HHE (with IVs they choose themselves), but also to recover some information from the server later on. Such a scenario is thus not precluded. For example, we can consider a malicious server using the client as a chosen ciphertext decryption/verification oracle. This could happen in a Bleichenbacher type scenario where a client sends back a bit depending on the success of plaintext parsing (the malicious server has access to a symmetric decryption oracle), or a scenario where the server has access to the public result of a computation (access to a homomorphic decryption oracle).

6.1 Known-IV attack

In this section, as in the case of the attacks described previously, the attacker has access to a long keystream generated from a single known-IV. However, she will only consider some useful positions in this keystream. This allows to restrict the equations considered in the system to particular equations involving nonlinearly only key elements in a proper subset of size N', where N' is comprised between $(r-1) \cdot t = 4t$ and N, of the set of all key elements, of size N. To produce a keystream element, the XOF selects $(r-1) \cdot t$ key nibbles at the input of the $t = 12$ applications of h. For instance, we could only keep equations where those key nibbles entering the h functions are all picked in the first half of the key register (see Figure 5).

Working under this principle, the number of variables in the linearized system is reduced to $\mu_{N'} = \binom{N'}{4}\rho + (N - N')$, since the $N - N'$ LSBs of the key elements

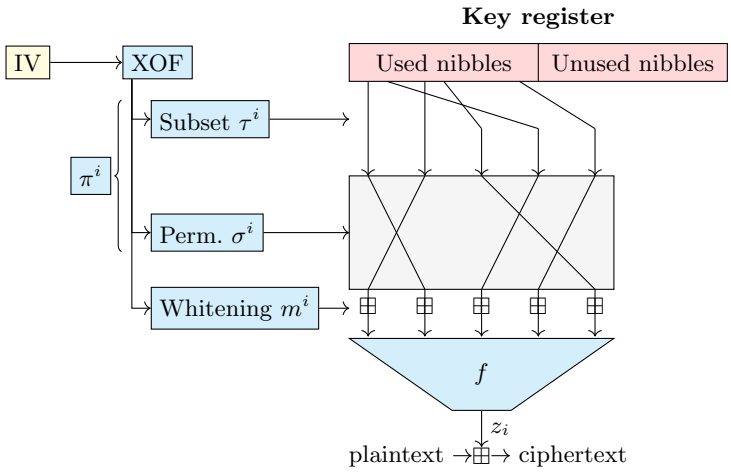

Figure 5. Selected equations chosen so that the inputs to the h functions correspond to key nibbles from the first half of the key register.

that appear only linearly through g still need to be considered. The number of particular equations to collect is of the same order $\delta_{N'} \gtrapprox \mu_{N'}$. Assuming that the XOF outputs are well distributed, the probability that all the key nibbles appearing at the input of the h functions belong to a subset of size N' is given by $p_{N'} = \binom{N'}{(r-1)\cdot t} / \binom{N}{(r-1)\cdot t}$. Thus the number of equations to obtain in order to get enough particular equations after filtering is $\delta_{N'}/p_{N'}$. This constitutes the data complexity of this attack. The time complexity of building the matrix $\mathbf{A}$ is now dominated by the cost of filtering the equations to keep only the particular equations. The time complexity is $\delta_{N'}/p_{N'}$ generations of parameters by the XOF. Then, the time complexity of the Block Wiedemann algorithm equals $\frac{6}{l_1} \cdot s \cdot \delta_{N'}^2$ operations, where 6 and l_1 are the same constants as in the previous Section 5.2. This enables the attacker to recover the N' key elements and the $N - N'$ LSBs of the other key elements using the previously described attack. Recovering the remaining $3(N-N')$ bits is an easier problem which can be linearized at a smaller cost.

Using the parameters from `Elisabeth-4`, we choose the size N' of the target subset so that the time complexity of the system construction and the time complexity of its resolution are similar. We find $N' = 137$, which makes the data complexity about 2^{87} nibbles and the time complexity about $2 \times 2^{87} = 2^{88}$ operations. Finally, the memory complexity is also improved, as storing $\mathbf{A}$ only requires about $t \cdot \rho \cdot \delta_{N'} = 2^{54}$ bits.

6.2 Chosen-IV attack

We now consider a chosen-IV version of this attack. In order to reduce the data complexity of the attack described in the previous subsection where the attacker had access to a large keystream (e.g., generated from a single known IV), we assume that the adversary can obtain the first nibble of the keystream for a large number of IVs of her choice.

Indeed, the attacker can prepare in an offline precomputation phase several well-chosen IV values such that the XOF selects in a predefined proper subset of the key register of size $N' < N$ the $(r-1) \cdot t$ key elements at the input of h in the computation of the first keystream nibble.

The precomputation of the well-chosen IVs costs $\delta_{N'}/p_{N'}$ generations of parameters by the XOF and enables to determine $\delta_{N'}$ appropriate IVs. The equations corresponding to the first element of the keystream generated from these IVs enable to build the matrix $\mathbf{A}$ for a data complexity of $\delta_{N'}$.

We optimize the choice of N' so as to ensure that the time complexity of the IV precomputation equals the time complexity of the key-recovery step. This results in the same optimal value $N' = 137$ as in the known-IV setting. The time complexity of the IV precomputation is then 2^{87} generations of parameters by the XOF, the data complexity is only $\mu_{N'} = 2^{37}$ elements, the time complexity of the construction and resolution of the linear system is 2^{87} elementary operations. The memory complexity of the attack is about $t \cdot \rho \cdot \delta_{N'} = 2^{54}$ bits.

7 Small-scale experiments

In order to illustrate the linearization attacks presented in this paper, and to validate their correctness, we implemented an attack on a reduced version of `Elisabeth-4`. In this section, we describe the reduced version we considered for our tests and report on the results we obtained. The implementation of our experiments can be found at the following address:

`https://github.com/jj-anssi/asiacrypt2023-cryptanalysis-elisabeth4`

7.1 `Elisabeth-4` reduced version

`Elisabeth-4` design is highly parameterizable, which enables to easily define reduced versions. In order to implement practically the attacks presented in this paper, we need to reduce the size of the cipher, but we want to do so in a way that remains representative of the structure of the cipher. We emphasize that we not make any claim of resistance of the reduced version of `Elisabeth-4` considered here against any kind of attack.

In order to retain the effects of the rank defect discussed in Section 4, we choose not to modify the structure of the function g. Also, we require $t \geq 2$, since there is some decoupling in the linear system between the components related to different key quartets when $t = 1$. Consequently, we need $N \geq t \cdot r = 10$. Since the memory size of the sparse matrix $\mathbf{A}$ is already quite large at this point, namely

$\binom{10}{4} \cdot t \cdot \rho^2 \approx 2^{35}$, we reduce further the design by considering 3-bit elements. Our reduced variants thus operates in $\mathbb{Z}_8$, and eight arbitrary negacyclic Sboxes are selected (see Table 2). We continue to observe a rank defect in this case with $\rho = 254 \ll 2^{12}$. This allows to increase the number N of key elements and the number t of parallel calls to g in the filtering function f.

Table 2. Comparison of the parameters of the original `Elisabeth-4` cipher and the toy cipher introduced in this paper for experimental verifications.

`Elisabeth-4`	N	t	r	Sboxes	Group	ρ
Original	256	12	5	4-bit	$\mathbb{Z}_{16}$	8705
Toy	32	2	5	3-bit	$\mathbb{Z}_8$	254

While implementing this reduced version, we were under the impression that the XOF of `Elisabeth-4` is not precisely defined in [3]. It refers to [13] as a source for the definition of a *forward secure PRG*, but the correspondence from random bits to random integers encoded in a function `next_int` [3, Algorithm 2] is not clearly explicited. Furthermore, the Rust implementation of `Elisabeth-4`[11] directly uses a `RandomGenerator` object from the `concrete-core` Rust library. In order to ensure interoperability, the XOF function should be explicitly and unambiguously defined. In our implementation, we try to fill the gaps in a sensible way. For more details on how we defined the XOF in our implementation, we refer the reader to Annex E and to our code (of which the link can be found at the beginning of this section). Please note that our cryptanalysis does not depend on the details of the XOF and of the functions that produce the round parameters of Elisabeth-4 from the output of the XOF.

7.2 Implemented attack

We have implemented the linearization attack, taking into account the analysis performed in Sections 4 and 5.

In order to implement the resolution of the sparse linear system, we used the implementation of the Block Wiedemann algorithm provided by the CADO-NFS project [4]. CADO-NFS is an open-source tool enabling to factor large integers using the state-of-the-art factorization algorithms [4]. It has been used to achieve factorization records in past years. A resource intensive phase of the factorization algorithm consists in finding a vector in the kernel of a large sparse matrix, thus CADO-NFS provides a state-of-the-art, parallelisable and distributable implementation of the Block Wiedemann algorithm, `linalg/bwc`. This implementation routinely solves linear algebra problems with sparse matrices with millions of rows and about a hundred nonzero entries per rows.

[11] `https://github.com/princess-elisabeth/Elisabeth`.

During our experiments, we encountered issues using `linalg/bwc`. We realized that this implementation is tailored to the case of matrices encountered in the context of the NFS algorithm. More precisely, `linalg/bwc` solves $\mathbf{x}.\mathbf{A} = \mathbf{0}$, with matrix $\mathbf{A}$ whose number of rows exceeds its number of columns, whereas considering an overdetermined system $\mathbf{A}$ places us in the case where the number or rows is less that the number of columns. There seems to be some issue with the padding of matrix $\mathbf{A}$ to a square matrix, since the `gather` program which is part of `linalg/bwc` reports nonzero coordinates in the padding part. In order to bypass these issues, we fallback to generating square $\mathbf{A}$ matrices, embedding the vector $\mathbf{z}$ as a column in the matrix $\mathbf{A}$. In other words, we are using exactly $\delta = \mu + 1$. This constraint is not prohibitive as confirmed by the experimental results below. Further investigations of the encountered issues in `linalg/bwc` and solving them in order to get a general purpose implementation of the Block Wiedemann algorithm is left as an open problem.

Results. The goal of our experiment was to check that the improved attack of Section 5.2 works rather than experimentally evaluating its complexity. Our experiment showed that it is possible to solve an Elisabeth-4 type linear system using BW algorithm and that solving this linear system allows to recover the secret key. Indeed, we managed to run the linearization attack successfully on a reduced version of `Elisabeth-4` (with 3-bit Sboxes, $N = 32$ and $t = 2$), and recovered the key from a found kernel vector. This result gives some confidence in the correctness of the attack.

As detailed in Section 5.2, the most expensive part of the attack performed is the Block-Wiedemann step. Its theoretical complexity is $6 \cdot t \cdot \binom{N}{4}^2 \cdot \rho^3 / l_1$. As in our experiment, $t = 2$, $N = 32$, $\rho = 254$, $l_1 = 64$, we obtain a theoretical complexity of $2^{51.8}$ elementary operations. On the multicore platform that was used for the experiment, the attack required 44 hours. Yet, we highlight that, due to the impact of an increased memory complexity, it is difficult to assess how these results scale when considering larger instances.

Acknowledgement. This work is partially supported by the French Agence Nationale de la Recherche through the SWAP project under Contract ANR-21-CE39-0012.

References

1. Chillotti, I., Joye, M., Paillier, P.: Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In: Dolev, S., Margalit, O., Pinkas, B., Schwarzmann, A.A. (eds.) Cyber Security Cryptography and Machine Learning - 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8-9, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12716, pp. 1–19. Springer (2021). https://doi.org/10.1007/978-3-030-78086-9_1, https://doi.org/10.1007/978-3-030-78086-9_1

2. Coppersmith, D.: Solving homogeneous linear equations over gf(2) via block wiedemann algorithm. Mathematics of Computation **62**(205), 333–350 (1994), http://www.jstor.org/stable/2153413
3. Cosseron, O., Hoffmann, C., Méaux, P., Standaert, F.X.: Towards Globally Optimized Hybrid Homomorphic Encryption - Featuring the Elisabeth Stream Cipher. In: ASIACRYPT 2022. Taipei, Taiwan (2022), https://hal.inria.fr/hal-03905546
4. CADO-NFS Development Team, T.: CADO-NFS, an implementation of the number field sieve algorithm (2017), http://cado-nfs.inria.fr/, release 2.3.0
5. Duval, S., Lallemand, V., Rotella, Y.: Cryptanalysis of the FLIP family of stream ciphers. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology – CRYPTO 2016, Part I. Lecture Notes in Computer Science, vol. 9814, pp. 457–475. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2016). https://doi.org/10.1007/978-3-662-53018-4_17
6. Faugère, J.C.: 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. p. 75–83. ISSAC '02, Association for Computing Machinery, New York, NY, USA (2002). https://doi.org/10.1145/780506.780516, https://doi.org/10.1145/780506.780516
7. Faugére, J.C.: A new efficient algorithm for computing gröbner bases (f4). Journal of Pure and Applied Algebra **139**(1), 61–88 (1999). https://doi.org/https://doi.org/10.1016/S0022-4049(99)00005-5, https://www.sciencedirect.com/science/article/pii/S0022404999000055
8. Joux, A.: Algorithmic Cryptanalysis. Chapman & Hall/CRC Cryptography and Network Security Series, Taylor & Francis (2009), https://books.google.fr/books?id=dyavmAEACAAJ
9. Joux, A., Pierrot, C.: Nearly sparse linear algebra and application to discrete logarithms computations. In: Contemporary Developments in Finite Fields and Applications, pp. 119–144. World Scientific (2016)
10. Kaltofen, E.: Analysis of coppersmith's block wiedemann algorithm for the parallel solution of sparse linear systems. In: Cohen, G., Mora, T., Moreno, O. (eds.) Applied Algebra, Algebraic Algorithms and Error-Correcting Codes. pp. 195–212. Springer Berlin Heidelberg, Berlin, Heidelberg (1993)
11. Massey, J.: Shift-register synthesis and bch decoding. IEEE transactions on Information Theory **15**(1), 122–127 (1969)
12. Méaux, P., Carlet, C., Journault, A., Standaert, F.X.: Improved filter permutators for efficient FHE: Better instances and implementations. In: Hao, F., Ruj, S., Sen Gupta, S. (eds.) Progress in Cryptology - INDOCRYPT 2019: 20th International Conference in Cryptology in India. Lecture Notes in Computer Science, vol. 11898, pp. 68–91. Springer, Heidelberg, Germany, Hyderabad, India (Dec 15–18, 2019). https://doi.org/10.1007/978-3-030-35423-7_4
13. Méaux, P., Journault, A., Standaert, F.X., Carlet, C.: Towards stream ciphers for efficient FHE with low-noise ciphertexts. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016, Part I. Lecture Notes in Computer Science, vol. 9665, pp. 311–343. Springer, Heidelberg, Germany, Vienna, Austria (May 8–12, 2016). https://doi.org/10.1007/978-3-662-49890-3_13
14. Strassen, V.: Gaussian elimination is not optimal. Numerische Mathematik **13**, 354–356 (1969)
15. Wiedemann, D.: Solving sparse linear equations over finite fields. IEEE transactions on information theory **32**(1), 54–62 (1986)

A Sboxes

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
S_1	3	2	6	c	a	0	1	b	d	e	a	4	6	0	f	5
S_2	4	b	4	4	4	f	9	c	c	5	c	c	c	1	7	4
S_3	b	a	c	2	2	b	d	e	5	6	4	e	e	5	3	2
S_4	5	9	d	2	b	a	c	5	b	7	3	e	5	6	4	b
S_5	3	0	b	8	d	e	d	b	d	0	5	8	3	2	3	5
S_6	8	d	c	c	3	f	c	7	8	3	4	4	d	1	4	9
S_7	4	2	9	d	a	c	a	7	c	e	7	3	6	4	6	9
S_8	a	2	5	5	3	d	f	1	6	e	b	b	d	3	1	f

Table 3. Sboxes used in `Elisabeth-4` in hexadecimal notations.

B Degree of the least significant bit

In this section, we show that at any iteration j of the keystream generator, the ANF of the Boolean function f_j that takes as input the key bits and returns the least significant bit of the keystream element is bounded by 12. We remind the reader that this proof uses definitions and propositions introduced in Section 4.3.

We showed that at any iteration j, the Boolean function f_j can be written as the sum of $t = 12$ Boolean variations of h and the least significant bits of some key elements (see Section 3). Thus, the degree of f_j is bounded by the maximum degree of a variation of h. Further, in Section 4.3, we showed that any Boolean variation of h can be written as the sum of four Antler functions. Thus, the maximum degree of any variation of h, and thus the degree of f_j, is bounded by the maximum degree of an Antler function. The following Theorem and its proof thus suffice to upper bound the degree of f_j.

Theorem 1. *For any* 3 *negacyclic look-up tables* $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$*, the ANF of* $H_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}$ *has degree at most 12.*

Proof of Theorem 1. In order to study $H_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}$, we introduce the following definitions (see Figure 6):

$$\begin{aligned} u &\coloneqq \mathcal{S}_1[b] = \mathcal{S}_1[x+y] \\ v &\coloneqq \mathcal{S}_2[b'] = \mathcal{S}_2[y+z] \\ \beta &\coloneqq \mathcal{S}_1[b] + \mathcal{S}_2[b'] + w\,. \end{aligned}$$

Each bit of b (resp. b', $u+v$) can be expressed as a polynomial in the sum of the bits of x and y (resp. y and z, u and v) and in the product of the bits of x

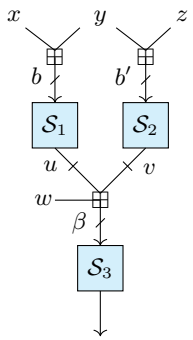

Figure 6. The Antler function $H_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}$.

and y (resp. y and z, u and v). This property reflects the symmetric nature of the modular addition. Thus, for $0 \leq i \leq 3$, we also define

$$\begin{aligned} s_i &\coloneqq x_i \oplus y_i, s'_i = y_i \oplus z_i \\ p_i &\coloneqq x_i \cdot y_i, s'_i = y_i \cdot z_i \\ \sigma_i &\coloneqq u_i \oplus v_i \\ \pi_i &\coloneqq u_i \cdot v_i \,. \end{aligned}$$

For any $0 \leq i \leq 3$, the following equations are verified

$$\begin{aligned} p_i s_i &= 0 \\ p'_i s'_i &= 0 \\ \pi_i \sigma_i &= 0 \,. \end{aligned}$$

As a consequence, by applying the definition of the modular addition to $b = x + y$, we get

$$\begin{aligned} b_0 &= s_0 \\ b_1 &= p_0 + s_1 \\ b_2 &= p_0 s_1 + p_1 + s_2 \\ b_3 &= p_0 s_1 s_2 + p_1 s_2 + p_2 + s_3 \,. \end{aligned}$$

The exact same equations hold for b'. Similarly, the bits of $u + v$ can be expressed as polynomials in the σ_i's and π_i's.

Going back to the proof, the main idea is to show that the monomials in x_i, y_i, z_i, w_i for $0 \leq i \leq 3$ that can appear in the ANF of $H_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}(x, y, z, w)$ have their degree bounded by 12. The first property we use is that $H_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}(x, y, z, t)$ does not depend on β_3. This is a direct consequence of Proposition 2. On the other hand, $H_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}(x, y, z, t)$ can be expressed as a sum of monomials in the β_i's for $i = 0, 1, 2$. In order to prove the theorem, we thus simply need to show that any monomial in $\beta_0, \beta_1, \beta_2$ can be expressed as a sum of monomials in x_i, y_i, z_i, w_i for $0 \leq i \leq 3$ that all have degree at most 12.

To do so, we first need to express β_0, β_1, β_2 as polynomials in the σ_i's, π_i's and the w_i's. By applying the modular addition to $\beta = u + v + w$, we obtain

$$\begin{aligned}\beta_0 &= w_0 + \sigma_0\\ \beta_1 &= w_1 + w_0\sigma_0 + \pi_0 + \sigma_1\\ \beta_2 &= w_2 + w_0w_1\sigma_0 + w_1(\pi_0 + \sigma_1) + w_0\sigma_0\sigma_1 + \pi_0\sigma_1 + \pi_1 + \sigma_2\,.\end{aligned}$$

Considering monomials in the β_i's for $i = 0, 1, 2$, we study which monomials in the σ_i's, π_i's and the w_i's can appear in their polynomial expression. For example, the monomial $\beta_0\beta_1$ can be expressed as

$$\beta_0\beta_1 = w_0w_1 + w_1\sigma_0 + w_0\pi_0 + w_0\sigma_1 + \sigma_0\sigma_1$$

and thus contains the monomials w_0w_1, $w_1\sigma_0$, $w_0\pi_0$, $w_0\sigma_1$ and $\sigma_0\sigma_1$. We show that the only monomial that can appear in the polynomial expression of a monomial in the β_i's, $0 \le i \le 2$ that depends on the three variables w_0, w_1 and w_2 is $w_0w_1w_2$. β_2 is the only variable that depends on w_2, β_0 does not depend on w_1 and β_1 depends only linearly on w_1. Thus, for a monomial in the three variables w_0, w_1 and w_2 to appear in the expression of a monomial in the β_i's, one must consider $\beta_0\beta_1\beta_2$, in which only the monomial $w_0w_1w_2$ depends on all three variables. Since only $w_0w_1w_2$ can appear, any monomial that depends on the σ_i's and π_i's depends on at most two of the w_i's. We will now show that the monomials in the σ_i's and π_i's expressed as polynomials in the x_i's, y_i's and z_i's are of degree at most 10, which will conclude the proof of the theorem.

The σ_i's and π_i's, as well as any monomial in these variables, can be expressed as a function of b and b'. In turn, b and b', as well as any monomial in these variables, can be expressed as a sum of monomials in the p_i's, s_i's, p'_i's and s'_i's. Since p_3 (resp. p'_3) does not appear in the expression of b_i (resp. b'_i), $0 \le i \le 3$, the monomials that can appear in the expression of a monomial in the σ_i's and π_i's do not depend on p_3. Further, recall that $p_is_i = p'_is'_i$. Thus, the monomials cannot depend on both p_i and s_i (resp. p'_i and s'_i). Last but not least, note that any $p_ip'_i = x_iy_iz_i$ is of degree 3. At first sight, the monomial of highest degree that can be formed is $s_3s'_3\prod_{i=0}^{2} p_ip'_i$. This monomial has degree 11, and is the only monomial of degree 11 that respects the constraints we have put forward. However, it turns out that this monomial cannot appear. Indeed, only b_3 (resp. b'_3) is the depends on s_3 and p_2 (resp. s'_3 and p'_2). Further, in the polynomial expression of b_3 (resp. b'_3), these variables are added to each other. It comes that in the polynomial expression of any monomial in the b_i's, these variables cannot be multiplied with each other. Thus, the monomials of highest degree that can be formed have degree 10. We have thus shown the theorem.

C Proof of Proposition 4

By Lemma 1, we only need showing that the rank of the set of functions $G_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}$ has its dimension bounded by $2^{10.43}$. It is straightforward that the vector space

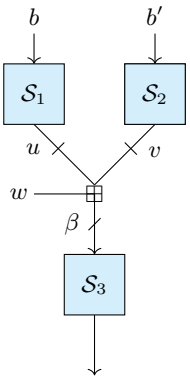

Figure 7. The function $G_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}$.

spanned by all functions $G_{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}$ is a linear subspace of a vector space of the form $B_0 + w_0B_1 + w_1B_2 + w_2B_3 + w_0w_1B_4 + w_0w_2B_5 + w_1w_2B_6 + w_0w_1w_2B_7$ where each B_i is a linear subspace of the linear space of dimension 2^8 spanned by the monomials in the bits of b and b'. The dimension of this vector space can thus be upper bounded by the sum $\sum_{i=0}^{7} \dim(B_i)$. We provide an upper bound on $\dim(B_i)$ for each i. We remind the reader of the following notations (see Figure 7):

$$\begin{aligned} u &\coloneqq \mathcal{S}_1[b] \\ v &\coloneqq \mathcal{S}_2[b'] \\ \beta &\coloneqq \mathcal{S}_1[b] + \mathcal{S}_2[b'] + w \\ \sigma_i &\coloneqq u_i \oplus v_i \\ \pi_i &\coloneqq u_i \cdot v_i \,. \end{aligned}$$

We also remind the reader of the expression of the value of the bits β_0, β_1 and β_2 as polynomials in the σ_i's, π_i's and w_i's.

$$\begin{aligned} \beta_0 &= w_0 + \sigma_0 \\ \beta_1 &= w_1 + w_0\sigma_0 + \pi_0 + \sigma_1 \\ \beta_2 &= w_2 + w_0w_1\sigma_0 + w_1(\pi_0 + \sigma_1) + w_0\sigma_0\sigma_1 + \pi_0\sigma_1 + \pi_1 + \sigma_2 \,. \end{aligned}$$

The least significant bit of h is a linear combination of monomials in β_i for $i = 0, 1, 2$. First, we consider B_7, which corresponds to $w_0w_1w_2$. The only monomial in the β_i's that contains a monomial dividable by $w_0w_1w_2$ is $\beta_0\beta_1\beta_2$. Further, the only monomial that can appear is $w_0w_1w_2$. Thus, B_7 has dimension 1. Next, we consider B_6, which corresponds to w_1w_2. The only two monomials in the β_i's that contain monomials dividable by w_1w_2 are $\beta_1\beta_2$ and $\beta_0\beta_1\beta_2$. We now consider the monomials dividable by w_1w_2 but not dividable by w_0 that appear in $\beta_1\beta_2$ and $\beta_0\beta_1\beta_2$. We obtain the set $\{w_1w_2, w_1w_2\sigma_0\}$. By Proposition 2, σ_0 is the sum of two bits that depend on 3 bits each and thus the rank of the

set of σ_0 has rank $2^3 + 2^3 = 2^4$. It comes that B_6 has dimension at most 2^4. Lastly, we consider B_5, which corresponds to w_0w_2. Three monomials in the β_i's contain monomials dividable by w_0w_2, namely $\beta_0\beta_2$, $\beta_1\beta_2$ and $\beta_0\beta_1\beta_2$. We now consider the monomials dividable by w_0w_2 but not dividable by w_1 that appear in these three monomials. We obtain the set $\{w_0w_2, w_0w_2\sigma_0, w_0w_2\pi_0, w_0w_2\sigma_1\}$. By Proposition 2, the set of π_0 functions has rank at most $2^3 \times 2^3 = 2^6$. Further, it contains the linear subspace spanned by the σ_0 functions. The set of σ_1 functions has dimension $2^4 + 2^4 = 2^5$ and also contains the linear subspace spanned by the σ_0 functions. Thus, B_5 has dimension at most $2^6 + 2^5 - 2^4$. It comes that a bound on the rank of the LSB of h is $\sum_{i=0}^{7} \dim(B_i) \leq 5 \times 2^8 + 2^6 + 2^5 - 2^4 + 2^4 + 1 \approx 2^{10.43}$.

D On the dimension of the affine space of solutions

In this section, we discuss the dimension of the affine space of solutions to the matrix equation $\mathbf{Ax} = \mathbf{z}$ for a matrix $\mathbf{A}$ constructed as described in Section 4 and Section 6. In particular, $\mathbf{A}$ is a matrix of size $\delta \times \binom{N'}{4}\rho$ where $\delta \gtrapprox \binom{N'}{4}\rho$ and where $N' = N$ in Section Section 4 and $N' < N$ in Section Section 6. Ignoring a few lone nonzero bits from the final linear contribution to g, each row of $\mathbf{A}$ has less than $t \cdot \rho$ active bits, organized into $t = 12$ sets of ρ active bits. A necessary condition for the affine space of solutions to have dimension 1 is that each of the $\binom{N'}{4}$ submatrices of size $\delta \times \rho$ of $\mathbf{A}$, constructed by extracting the ρ columns corresponding to an unordered quartet of key element positions, is non-singular.

The concern that the matrix equation $\mathbf{Ax} = \mathbf{z}$ could have an affine space of solutions with a problematically large dimension, which would be highly unlikely for a random matrix, stems from the fact that $\mathbf{A}$ has the following structure: each row is nonzero in only t distinct sets of columns. If t was 'too' small, e.g. $t = 1$, then this necessary condition might not have been fulfilled: each submatrix would have on average just about ρ nonzero rows, and thus, the probability that all $\binom{N'}{4}$ submatrices have full rank would would be rather low. For $t = 12$, on the other hand, we provide a proof that this necessary condition is satisfied with overwhelming probability for the values N' we consider in our attacks. In other words, the structure of $\mathbf{A}$ produces no oblivious rank deficiency as compared with the behaviour of a random matrix.

We consider a submatrix of size $\delta \times \rho$ extracted from $\mathbf{A}$ as described above. We let p_1 be the probability that after gathering δ equations, this submatrix has less than $2 \cdot \rho$ nonzero rows. This probability is strictly smaller than the probability that after gathering $\binom{N'}{4}\rho =_{def} \delta_1 < \delta$ equations, this submatrix has less than $2 \cdot \rho$ nonzero rows. We compute this latter probability. For a fixed submatrice, we can view the construction of $\mathbf{A}$ as drawing $\delta_1 = \binom{N'}{4}\rho$ rows such that for each row, the probability that this row is nonzero is $p_t = t/\binom{N'}{4}$. The number X of nonzero rows thus follows a binomial law with parameters $\delta_1 = \binom{N'}{4} \cdot \rho$ and p_t, $X \sim B(\delta_1, p_t)$. Thus, the probability that after $\binom{N'}{4} \cdot \rho$ equations, the submatrix has less than $2 \cdot \rho$ nonzero rows is $\mathbb{P}(X \leq 2 \cdot \rho)$. Since $\delta_1 \cdot p_t \cdot (1 - p_t) \gg 10$, we use the approximation of the binomial distribution by the normal distribution

given by the Moivre-Laplace theorem:

$$\mathbb{P}(X \leq 2 \cdot \rho) \approx \mathbb{P}\left(Y \leq \frac{2 \cdot \rho - \delta_1 \cdot p_t}{\sqrt{\delta_1 \cdot p_t \cdot (1 - p_t)}}\right),$$

where $Y \sim \mathcal{N}(0, 1)$. For $N' = N = 256$ and $N' = 137$, this probability can be shown to be at most $\frac{e^{-269^2}}{538\sqrt{\pi}}$.

We now compute the probability p_2 that a submatrix containing at least $2 \cdot \rho$ equations does not have full rank ρ. We approximate this probability by the probability that a random $2 \cdot \rho \times \rho$ matrix does not have full rank. It can be shown by induction that the probability that a random $\mathbb{F}_2$-matrix of size $2 \cdot \rho \times \rho$ has full rank ρ is at least $e^{-\frac{\rho}{2\rho+1}}$. This implies that $p_2 \leq 1 - e^{-\frac{\rho}{2\rho+1}}$.

For a fixed submatrice, p_1 is the probability that after gathering δ equations, this submatrix has less than $2 \cdot \rho$ nonzero rows whilst p_2 is the probability that a submatrix containing at least $2 \cdot \rho$ equations does not have full rank. Thus, the probability that a fixed submatrix is singular is at most $p_1 + p_2$. Since there are $\binom{N'}{4}$ submatrices, the probability that at least one submatrix is singular is at most $\binom{N'}{4}(p_1 + p_2)$. In particular, for $N' = N = 256$ and $N' = 137$, each submatrix is non-singular with probability at least $1 - \binom{N'}{4}(p_1 + p_2) > 0.99$.

E Description of the XOF

The XOF state contains an AES key. It is initialized with the IV. During operation, a block of output is produced by encrypting a fixed constant using the key in the XOF state. The updated state is obtained by encrypting another fixed constant using the same XOF state as key. This enables to produce a sequence of bits of arbitrary length. From this sequence we extract bit sequences to generate masking values and integers to generate an ordered arrangement. In order to generate an integer uniformly at random in $\{0, .., n-1\}$, we apply rejection sampling. We form an integer from k bits of the XOF output, where k is the bitlength of n. While this candidate integer is greater or equal that n, we discard it and take a new candidate. In the other case we use this integer as the output. This defines a procedure $next\_int(n)$. Note that the state of the XOF is updated as bits of its output sequence are consumed.

Using this building block, we follow exactly Algorithm 2 of [3]. Note that the generation procedure of parameters π^i and m^i is stateful: a current permutation of $\{1, .., N\}$ and an array of N masking values is maintained. At every step, we need to extract $r \cdot t$ (= 60 for `Elisabeth-4`, = 10 in our toy version) key nibbles. We do so by performing an 'aborted Knuth shuffle': for i in $\{1, \cdots, r \cdot t\}$, we compose the current permutation with the transposition $(i, i + x)$ where x is an output of $next\_int(N - i)$. At the end of the loop, the $r \cdot t$ first positions of the current permutation contain an uniformly distributed ordered arrangement of $\{1, .., N\}$. For every position determined by this arrangement, we add in the array of masking values a fresh group element, generated through $next\_int(16)$.