IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2023, No. 3, pp. 97–135. DOI:10.46586/tches.v2023.i3.97-135

Cryptanalysis of ARX-based White-box Implementations*

Alex Biryukov, Baptiste Lambin and Aleksei Udovenko

University of Luxembourg, Esch-sur-Alzette, Luxembourg
firstname.lastname@uni.lu

Abstract. At CRYPTO'22, Ranea, Vandersmissen, and Preneel proposed a new way to design white-box implementations of ARX-based ciphers using so-called *implicit* functions and quadratic-affine encodings. They suggest the Speck block-cipher as an example target.
In this work, we describe practical attacks on the construction. For the implementation without one of the external encodings, we describe a simple algebraic key recovery attack. If both external encodings are used (the main scenario suggested by the authors), we propose optimization and inversion attacks, followed by our main result - a multiple-step round decomposition attack and a decomposition-based key recovery attack.
Our attacks only use the white-box round functions as oracles and do not rely on their description. We implemented and verified experimentally attacks on white-box instances of Speck-32/64 and Speck-64/128. We conclude that a single ARX-round is too weak to be used as a white-box round.

Keywords: White-box cryptography · Cryptanalysis · Algebraic attacks · Decomposition attacks

1 Introduction

Cryptanalysis of cryptographic primitives is usually done in the *black-box* model, where an attacker can query some oracle to obtain plaintext/ciphertext pairs, but without having direct access to the internal computations done by the oracle (still knowing which primitive is implemented obviously). While this model is nice from a theoretical point of view, in practice, cryptographic algorithms are deployed in a more hostile environment, where an attacker has some access to the hardware/software implementing these cryptographic primitives. This can for example lead to the attacker being able to examine execution time, power consumption, etc. which lead to *side-channel attacks* or the so-called *gray-box* attack model. However, one can go further and consider the *white-box* model, where the attacker actually has direct access and control to the (software) implementation of some cryptographic primitive, meaning that they can actually read and/or modify the code, make some partial executions, etc. This is a much stronger model, giving a lot of power to the attacker, but it remains relevant in contexts like DRM (Digital Rights Management) and mobile payments [AABM20, ABF+20].

Starting with the works of Chow, Eisen, Johnson and van Oorschot [CEJv03, CEJvO03] in 2002 (also called "the CEJO framework"), several proposals were made to give a white-box implementation of cryptographic primitives, typically of the AES and DES block ciphers [LN05, XL09, BCH16, RP20]. However, all of these proposals to "white-box" AES

*The work was supported by the Luxembourg National Research Fund's (FNR) and the German Research Foundation's (DFG) joint project APLICA (C19/IS/13641232).

Received: 2023-01-15 Accepted: 2023-03-15 Published: 2023-06-09

Table 1: Summary of attacks proposed in this paper. The branch size is denoted by n. Complexity is computed assuming encodings from [RVP22]. Target depends on round encodings: **A** for affine encodings, **Q** for sparse quadratic-affine encodings. Key recovery attacks were experimentally verified on instances with $n = 16, 32$. Optimization and inversion attacks were experimentally verified on affine-encoded instances. Time does not include the cost to compute the queries (data), which typically require *at least* $\mathcal{O}(n^3)$ time (depends on the implementation).

Attack	Target	Ref.	Time	Data	Comment
Algebraic key recovery	**A**	Sec.4	$\mathcal{O}(n^3)$	$\mathcal{O}(n)$	No external encoding (any side)
	Q	Sec.4	$\mathcal{O}(n^6)$	$\mathcal{O}(n^2)$	
Round oracle optimization	**A**	Sec.5.5	$\mathcal{O}(n^6)$	$\mathcal{O}(n^2)$	Computes bilinear implicit function
	Q	Sec.5.5	$\mathcal{O}(n^9)$	$\mathcal{O}(n^3)$	Computes quadratic-affine implicit function
Round oracle inversion	**A**	Sec.5.5	$\mathcal{O}(n^6)$	$\mathcal{O}(n^2)$	Requires existence of bilinear implicit function
	Q	Sec.5.5	$\mathcal{O}(n^6)$	$\mathcal{O}(n^2)$	Heuristic, several bits have to be guessed
Round decomposition	**A**	Sec.6	$\mathcal{O}(n^4)$	$\mathcal{O}(n^3)$	Total query time is at least $\mathcal{O}(n^6)$, dominating the time complexity.
	Q	Sec.7	$\mathcal{O}(n^6)$	$\mathcal{O}(n^2)$	Recovers quadratic encoding.
Decomposition-based key recovery	**AQ**	Sec.8	$\mathcal{O}(n^6)$	$\mathcal{O}(n^3)$	Requires several consecutive decomposed rounds to recover the master key.

ended up falling to practical attacks [BGEC04, MGH09], and thus the problem of providing a secure white-box implementation of AES remains open.

Observing that proposals for an AES white-box failed, Ranea, Vandersmissen, and Preneel [VRP22, RVP22] looked at providing a white-box implementation of a totally different structure, namely the Speck cipher, which is an ARX Feistel Network. In [VRP22], the authors first protected the implementation using affine self-equivalences of modular addition but showed that this approach is insecure. The consequent design in [RVP22] relies on the use of so-called implicit functions, which describe the graph of the function in an implicit way, allowing the computation of a function by solving of a linear system of equations. This allows to use affine-quadratic self-equivalences of the modular addition in order to create quadratic encodings of the round inputs, which is a much stronger protection than affine self-equivalences. They also insist on the necessity of using *external* encodings, which are "simple" functions composed with the implementation, adding more security at the cost of altering the functionality. As their construction is rather different that the earlier white-box AES proposals, previous white-box cryptanalysis techniques are rather hard if not impossible to apply to their scheme.

Our contribution In this paper, we describe several attacks on their construction. Our attacks are summarized in Table 1. Proof-of-concept implementation is available at

https://github.com/cryptolu/implicit_ARX_whitebox_cryptanalysis

First, we consider the case when one of the external encodings is omitted. Although the authors or [RVP22] insist that external encodings are crucial, they also suggest that even

without external encodings their implementations are less vulnerable than, for example, designs of [CEJv03,RP20]. Furthermore, it is important to evaluate whether the new design contributes towards security against generic attacks such as the differential computation analysis (DCA) [BHMT16] or the algebraic attack (also known as Linear Decoding Analysis, LDA) [GPRW20,BU18], which are only applicable in the "pure" white-box setting without external encodings. In this setting, we show a practical degree-2 algebraic key-recovery attack on the white-box implementation of Speck proposed in [RVP22]. To the best of our knowledge, this is the first application of the algebraic attack to an ARX primitive.

Then, we focus on the setting with both external encodings. Typical attacks in this setting focus on decomposing round functions and analyzing interactions between them in order to recover the key (for example, the BGE attack [BGEC04] on the original proposal in [CEJv03]). Here, we describe algebraic and differential tools for analyzing the modular addition, including an algorithm for affine-equivalence of a quadratic Boolean function to a sum of a few monomials, black-box relation interpolation. We then show how algebraic relations of modular addition (described for example in [CD08]) can be used to optimize a white-box round oracle (effectively stripping the obfuscation of its implicit function by high-degree graph automorphisms), or to invert it (at a certain cost). We move on to the main result of our work - practical decomposition and key recovery attacks. We start by developing a decomposition method for an affine-encoded modular addition. We then show how a quadratic-affine encoding can be decomposed and reduced to the affine case. Finally, we show how to use round decompositions to perform full key recovery.

Our attacks were implemented and verified in practice on white-box instances of Speck32-64 and Speck64-128 generated by the code provided by the authors of [RVP22].

We remark that we only attack ARX-based white-box implementations, the implicit functions framework itself is still an interesting design tool (although our optimization and inversion techniques are quite generic and potentially threaten other round functions). Yet, it requires a round function with a simple quasilinear implicit function, a large set of quadratic-affine self-equivalences and graph automorphisms. So far, there are no such known round function candidates other than an ARX round.

2 Preliminaries and Notations

We use the 0-based big-endian notation for words, namely $x = (x_{n-1}, \dots, x_0)$, where x_{n-1} is the most significant bit and x_0 is the least significant bit. The i-th unit vector e_i is such that it's i-indexed bit is equal to one and other bits are equal to zero (its dimension should be clear from the context). The inner product between two n-bit vectors x, y is denoted by $\langle x, y \rangle = \sum_{i=0}^{n-1} x_i y_i$.

We focus on two-branch ARX-based implementations. We use n to denote the word size and $N = 2n$ to denote the block size. The addition modulo 2^n is denoted by $\boxplus$, the addition in $\mathbb{F}_2^n$ (XOR) is denoted by $\oplus$ or just $+$. We call the mapping

$$S : (\mathbb{F}_2^n)^2 \to (\mathbb{F}_2^n)^2 : (x, y) \mapsto (x \boxplus y, y)$$

a *bijective modular addition* (called "permuted addition" in [RVP22], which we find ambiguous). A white-box round oracle or its modification is denoted by $\mathbb{O} : \mathbb{F}_2^N \to \mathbb{F}_2^N$. The notation $\Delta\mathbb{O}$ denotes a randomized map $\mathbb{F}_2^N \to \mathbb{F}_2^N$, mapping a given $\Delta X \in \mathbb{F}_2^N$ to $\mathbb{O}(x) \oplus \mathbb{O}(x \oplus \Delta X)$ for a fresh random $x \xleftarrow{\$} \mathbb{F}_2^N$.

Each Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ can be expressed in the algebraic normal form (ANF): $f(x) = \sum_{u \in \mathbb{F}_2^n} \lambda_u(x)$, where $\lambda_u \in \mathbb{F}_2$ are the ANF coefficients. The algebraic degree of f is the maximum Hamming weight of u with $\lambda_u = 1$. The algebraic degree of $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is defined to be the maximum algebraic degree amount its output coordinates.

3 Implicit Encodings of ARX Primitives

Let $E_k = E_k^{(r)} \circ E_k^{(r-1)} \circ \cdots \circ E_k^{(1)}$ be an encryption function, where each $E_k^{(i)}$ represents the round function, depending on the key k in some way. For better clarity, we will drop the k from the notation of the round functions $E^{(i)}$. The main goal of white-box encryption is to provide *encoded* round functions $\overline{E}^{(i)}$ such that one cannot recover information about the key when only provided access to each $\overline{E}^{(i)}$, while also preserving the property that

$$\overline{E}^{(r)} \circ \overline{E}^{(r-1)} \circ \cdots \circ \overline{E}^{(1)} = E_k.$$

In practice, *external encodings* are necessary for security reasons, and thus the resulting white-box implementation would not be functionally equivalent to E_k, but to an encoded version of E_k, i.e.

$$\overline{E}^{(r)} \circ \overline{E}^{(r-1)} \circ \cdots \circ \overline{E}^{(1)} = O_{\text{ext}} \circ E_k \circ I_{\text{ext}},$$

where O_{ext} and I_{ext} are the external encodings, and depend on how the encoded round functions $\overline{E}^{(i)}$ are built.

In practice, these encoded round function $\overline{E}^{(i)}$ are built as $O^{(i)} \circ E^{(i)} \circ I^{(i)}$ where $I^{(i)}$ (resp. $O^{(i)}$) is the input (resp. output) encoding, and a white-box proposal describes how to build these encodings, as well as how to implement the resulting encoded round function. So far, to satisfy the property that the resulting implementation must result in the original encryption function, encodings were built such that $I^{(i+1)} = \left(O^{(i)}\right)^{-1}$, thus encodings in consecutive rounds cancel out and we only end up with $I^{(1)}$ and $O^{(r)}$ as the input and output external encodings, respectively. This ends up leading to the cancellation rule

$$\overline{E}^{(i+1)} \circ \overline{E}^{(i)} = O^{(i+1)} \circ E^{(i+1)} \circ E^{(i)} \circ I^{(i)}$$

for two consecutive rounds. However, the authors of [RVP22] proposed a way to generate encoded round functions satisfying the cancellation rule over 3 rounds (and not over 2 rounds):

$$\overline{E}^{(i+1)} \circ \overline{E}^{(i)} \circ \overline{E}^{(i-1)} = O^{(i+1)} \circ E^{(i+1)} \circ E^{(i)} \circ E^{(i-1)} \circ I^{(i-1)}$$

Their method is described below.

3.1 Description

The proposal from [RVP22] mainly relies on the notion of self-equivalence to provide encoded round functions.

Definition 1. For a function F, we say that a pair of invertible functions (A, B) is a self-equivalence of F if we have $F = B \circ F \circ A$. If A and B are linear (resp. affine), we say that (A, B) is a linear (resp. affine) self-equivalence. If A is affine and B is quadratic, we say that (A, B) is an affine-quadratic self-equivalence.

In the context of white-box encryption for ARX ciphers, the authors focus on the bijective modular addition, i.e. $F(x, y) = S(x, y) = (x \boxplus y, y)$, which is at the core of many ARX ciphers, and provide a way to obtain affine-quadratic self-equivalences for this function. By doing so, for a round function $E^{(i)} = L^{(i)} \circ S$ where S is the bijective modular addition and $L^{(i)}$ is an cipher's affine mapping containing the round-key, they generate encoded round functions as follows.

For each round function $E^{(i)}$, one picks a random invertible affine mapping $C^{(i+1)}$ as well as an affine-quadratic self-equivalence $(A^{(i)}, B^{(i)})$ of $E^{(i)}$, and the encodings for the round function $E^{(i)}$ are defined as

$$(I^{(i)}, O^{(i)}) = \left(A^{(i)} \circ B^{(i-1)} \circ (C^{(i)})^{-1}, C^{(i+1)}\right).$$

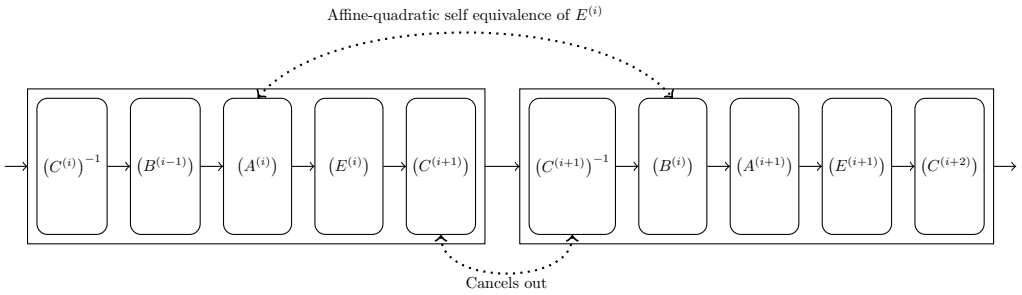

Figure 1: Cancellation of encodings for two consecutive rounds [RVP22]

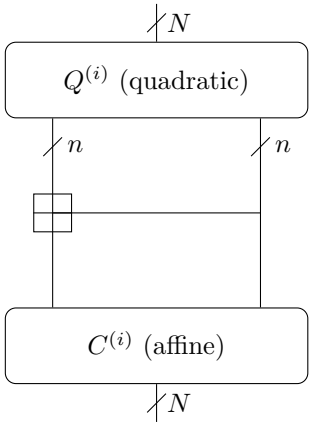

Figure 2: Internal structure of a single white-box ARX round with quadratic-affine encodings in the framework of [RVP22]. The quadratic encoding $Q^{(i)}$ is composed as $Q^{(i)} = A^{(i)} \circ L^{(i-1)} \circ B^{(i-1)} \circ (C^{(i)})^{-1}$, and the encoding $C^{(i)}$ is affine.

For external encodings, $I^{(1)}$ is built as $A^{(1)} \circ B^{(0)}$ with $A^{(1)}$ defined as above and $B^{(0)}$ a random invertible quadratic mapping[1], and $O^{(r)} = C^{(r+1)}$ with $C^{(r+1)}$ a random invertible affine mapping. Especially, the input encoding of a round is quadratic, while the output encoding is affine. Notice that these encodings do not satisfy $I^{(i+1)} = \left(O^{(i)}\right)^{-1}$, however they still allow to obtain an encoded encryption function, using the fact that

$$E^{(i)} = B^{(i)} \circ \left(C^{(i+1)}\right)^{-1} \circ C^{(i+1)} \circ E^{(i)} \circ A^{(i)}.$$

Figure 1 illustrates how the encodings of 2 consecutive rounds cancel each others and Figure 2 depicts the structure of one white-box round, which is one of our main attack targets.

Note that this gives the construction of the encoded round functions, but not their implementation. Indeed, one still needs an efficient way to evaluate these encoded round functions, without exposing the key material "hidden" inside. To this end, the authors of [RVP22] used the concept of *implicit functions* and *quasilinear functions*.

Definition 2 ([RVP22])**.** Let F be an n-bit function. A $(2n, m)$-bit function P is called an implicit function of F if it satisfies

$$P(x, y) = 0 \Leftrightarrow y = F(x)$$

[1]The authors of [RVP22] do not propose a method of sampling a random invertible quadratic mapping, and their implementation does not perform it as well.

Definition 3 ([RVP22])**.** A $(2n, m)$-bit implicit function P is quasilinear if for all $x \in \mathbb{F}_2^n$, the (n, m)-bit function $y \mapsto P(x, y)$ is affine.

Thus, to obtain an efficient implementation of the encoded round function $\overline{E}^{(i)}$, the authors of [RVP22] show a way to compute a quasilinear implicit function of $\overline{E}^{(i)}$. Then, for a given input value x, since the function $y \mapsto P(x, y)$ is affine, one simply solve the resulting affine system $P(x, y) = 0$ to compute the corresponding $y = \overline{E}^{(i)}(x)$. We refer the reader to [RVP22] for more details on how these quasilinear implicit functions are computed.

In Appendix A, we describe technical details on how we used the implementation of [RVP22] to generate instances of the Speck block cipher that we attacked.

4 Algebraic Cryptanalysis of Implicit White-box ARX Schemes with only one External Encoding

In [RVP22] in Section 4.2, the author made the conjecture that their implicit framework may be less vulnerable than previous constructions when the external encodings are trivial, with the following statement:

> While not the focus of this work, it is worth mentioning that this type of implicit implementations with trivial external encodings seems less vulnerable than CEJO or self-equivalences implementations with trivial external encodings.

In this section, we will show that without external encodings (more specifically, without either the input or the output external encoding), we can easily recover the round keys, and thus deduce the master key. Note that this is similar to the algebraic attack of [BU18, GPRW20] against implementations protected by a linear masking scheme. We start by explaining in details how to attack this construction when the *output* external encoding is trivial (i.e. no output external encoding), and briefly show how to do a similar attack when the input external encoding is trivial later. In essence, both attacks analyze intermediate states between rounds (which are protected by secret encodings) occurring in encryptions of random plaintexts. This bears similarity to gray-box-style differential computational analysis [BHMT16], with the difference that we perform algebraic analysis instead of correlation analysis.

4.1 When the output external encoding is trivial

Using the construction presented in the previous section, in general we end up with an encoded version of the encryption function E, that is $\overline{E} = O_{ex} \circ E \circ I_{ex}$ where O_{ex} and I_{ex} are the external encodings. Considering the case where we have no output external encodings means that when generating the encodings, we want to enforce that O_{ex} is the identity function. Taking a closer look at the encoded encryption function, we can observe that we get

$$\overline{E} = O_{ex} \circ E \circ I_{ex} = \left(C^{(r+1)} \circ \left(B^{(r)}\right)^{-1}\right) \circ E \circ \left(B^{(0)} \circ \left(C^{(1)}\right)^{-1}\right).$$

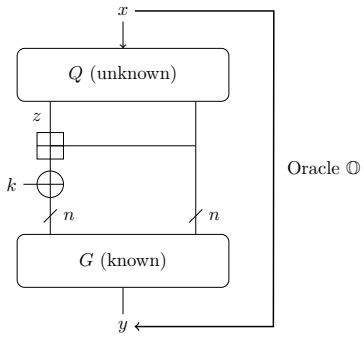

Figure 3: Structure of the function where we aim at recovering k in the case with no *output* external encoding

Thus we want $O_{ex} = \left(C^{(r+1)} \circ \left(B^{(r)}\right)^{-1}\right) = Id$, and we can deduce that the last round function is of the form

$$\begin{aligned}
\overline{E}^{(r)} &= C^{(r+1)} \circ E^{(r)} \circ A^{(r)} \circ B^{(r-1)} \circ \left(C^{(r)}\right)^{-1} \\
&= C^{(r+1)} \circ \left(B^{(r)}\right)^{-1} \circ B^{(r)} \circ E^{(r)} \circ A^{(r)} \circ B^{(r-1)} \circ \left(C^{(r)}\right)^{-1} \\
&= B^{(r)} \circ E^{(r)} \circ A^{(r)} \circ B^{(r-1)} \circ \left(C^{(r)}\right)^{-1} \quad \text{since } C^{(r+1)} \circ \left(B^{(r)}\right)^{-1} = Id \\
&= E^{(r)} \circ B^{(r-1)} \circ \left(C^{(r)}\right)^{-1} \quad \text{since } B^{(r)} \circ E^{(r)} \circ A^{(r)} = E^{(r)} \\
&= E^{(r)} \circ Q
\end{aligned}$$

where Q is some quadratic function, unknown from the attacker. Moreover, since we are in the context of white-box cryptography, we can choose any x and query the value of $F(x) = \overline{E}^{(r)}(x) = \left(E^{(r)} \circ Q\right)(x)$. Note that $E^{(r)}$ still depends on some (unknown) round key, which we aim at recovering. Focusing on the case of white-box implementations of the Speck cipher, we can further write this function with the structure depicted in Figure 3, where G is an invertible function known by the attacker (in this specific case for Speck, only an XOR from the left branch to the right branch, and a bitwise rotation of the right branch), and we aim at recovering k.

Now assume that we are somehow able to recover the round key in $E^{(r)}$ (which we will show next), it turns out that recovering the round key of the previous round $E^{(r-1)}$ leads to a very similar structure. Indeed, knowing the key used in $E^{(r)}$, we can now compute $\left(E^{(r)}\right)^{-1}(x)$ for any x of our choosing. Moreover, using the cancellation between encodings, notice that

$$\overline{E}^{(r)} \circ \overline{E}^{(r-1)} = E^{(r)} \circ E^{(r-1)} \circ Q'$$

for some quadratic function Q'. Since we know the key used in $E^{(r)}$, the function $F' = \overline{E}^{(r)} \circ \overline{E}^{(r-1)}$ can also be represented with the structure in Figure 3, and assuming we can recover the key in $E^{(r-1)}$, we can continue to go further and further until we recovered enough key information. Thus, our main focus will be to show how to recover the key from a function F with the structure given in Figure 3.

The general idea is that for a given input $x = (x_{N-1}, \ldots, x_1, x_0)$ (x_0 being the LSB), we can write the vector

$$\tilde{x} = (x_0x_1, x_0x_2, \ldots, x_0x_{N-1}, x_1x_2, \ldots, x_0, x_1, \ldots, x_{N-1}, 1)$$

i.e. a vector of length $L = N(N-1)/2 + N + 1$ containing all products between 2 coordinates as well as all coordinates themselves (and the constant 1). Then, since Q is a degree 2 function, $Q(x)$ can be computed as some (unknown) linear combinations of $\tilde{x}$, i.e. $Q(x) = M\tilde{x}$ for some unknown matrix M. Note that this matrix M is the same for any input x, since it essentially defines the ANF of Q. Knowing this, recovering k is a rather simple process. One first generate $L + \epsilon$ inputs x^i, compute their corresponding $\tilde{x}^i$ as well as $y^i = G^{-1}(F(x^i))$ (since G is known). By guessing the first (i.e. lowest significant) bit k_0 of k, we can then compute z_1^i (lowest significant bit of the left half of $Q(x^i)$) using y^i through the modular addition. According to our previous observation, if this key guess is correct, then for every i, we must have $z_0^i = M_0\tilde{x}^i$ for some $1 \times L$ matrix M_0 (which is the first row of M), i.e.

$$\begin{pmatrix} z_0^1 & z_0^2 & z_0^3 & \cdots & z_0^L \end{pmatrix} = M_0 . \begin{pmatrix} \tilde{x}^1 & \tilde{x}^2 & \tilde{x}^3 & \cdots & \tilde{x}^L \end{pmatrix}.$$

In other words, if the guess on k_0 is correct, the vector $z_0 = (z_0^1, z_0^2, z_0^3, \ldots, z_0^L)$ must belong to the column space of the matrix $\tilde{X}$ built with $\tilde{x}^i$ as columns, which can be efficiently done by precomputing a parity-check matrix for this space when generating the inputs x. Once the value of k_0 is determined, we can then do a similar process, guessing k_1 and using our knowledge of k_0 to compute z_1^i through modular substraction, and so on until we recovered k.

In practice, for some guesses on k_j, both values could lead to the vector z_j to belong to the column space of $\tilde{X}$. In this case, we just keep the resulting candidates and try each of them for the next guess. If one candidate for $k_j k_{j-1} \ldots k_0$ leads to z_{j+1} not being in the column space of $\tilde{X}$ for both values of k_{j+1}, then this candidate $k_j k_{j-1} \ldots k_0$ was actually incorrect and we can eliminate it. Thankfully, this behaves nicely for the modular addition, as after each guess of k_j, we are almost always left with only 2 candidates for the next guess.

We thus apply this procedure to $F = E^{(r)} \circ Q$, recovering (two) candidates for the round key used in $E^{(r)}$. Then for each of these candidates, we continue as described previously, now applying the procedure to $F' = E^{(r)} \circ E^{(r-1)} \circ Q'$ to recover candidates for the round key used in $E^{(r-1)}$. Again, in practice we get 2 candidates for the round key in $E^{(r)}$, but one of them gets eliminated when trying to recover the key in F'. Thus we end up with 2 candidates for the round key of $E^{(r-1)}$ (and now only one for $E^{(r)}$), and we can keep going back further and further into the rounds until we recovered enough key material to recover the master key. For Speck specifically, at worst the master key is of length $4n$, so we need to recover the round key for 4 consecutive rounds. Since we would end up with 2 candidates for the 4th round (starting from the end), we can apply this procedure for the 5 last rounds, thus uniquely recovering the value of the round keys used in the last 4 rounds, which allows us to determine the master key using the key-schedule.

Note that this last step is the only time where we use the key-schedule, so technically, if one were to use independent round keys instead, we could continue up to the very first round and thus obtain 2 candidates for the whole set of (independent) round keys, meaning that adding more rounds with independent round keys only increase the time complexity linearly with the number of rounds added.

The complexity of recovering candidates for the round key of a single round can be determined as follow. We first generate $L + \epsilon = \mathcal{O}(n^2)$ inputs x, compute the corresponding $\tilde{x}$ and generate a parity-check matrix H for the resulting space, which requires about $\mathcal{O}(n^6)$ operations. Note that this is only done once overall, even when recovering round keys on multiple rounds. Next we query the oracle to obtain $y = G^{-1}(F(x))$, so $\mathcal{O}(n^2)$ queries to the oracle and calls to G^{-1}. Then for each guess of k_j, we compute z_j and check if it belongs to the column space of $\tilde{X}$ using the parity check matrix H, i.e. computing the product $H \times z_j$ which takes $\mathcal{O}(n^4)$ operations, which we thus need to do at least n times (once for each bit of the key k), for a total of $\mathcal{O}(n^5)$ operations in this step. Note that

this assumes that we are only left with one candidate after each key guess. In practice, we are almost left with only 2 candidates after each guess, which only adds a constant factor. Thus overall, the complexity for one round is about $\mathcal{O}(n^6)$ bit-operations (i.e. linear algebra with matrices and vectors in $\mathbb{F}_2$) and $\mathcal{O}(n^2)$ calls to the oracle. From our experiments, the calls to the oracles are the dominant part of this algorithm, even using a pre-compiled shared C-library to make queries (which is much faster than using the native Python implementation given by the authors of [RVP22]). For example, for Speck32 (i.e. $n = 16$), to recover the round keys over the last 5 rounds (which allow to uniquely get the round keys of the last 4 rounds and thus recover the master key), our implementation in SageMath runs in about 79 seconds, but with 67 seconds spent in the oracle calls, thus only 12 seconds spent on actual computations.

4.2 When the input external encoding is trivial

We now consider the case where the input external encoding is trivial. Recall that our encoded cipher can be written as

$$\overline{E} = O_{ex} \circ E \circ I_{ex} = \left(C^{(r+1)} \circ \left(B^{(r)}\right)^{-1}\right) \circ E \circ \left(B^{(0)} \circ \left(C^{(1)}\right)^{-1}\right).$$

Thus if the input external encoding I_{ex} is trivial, we have $B^{(0)} \circ \left(C^{(1)}\right)^{-1} = Id$. Taking a closer look at the first encoded round function, we can see that is has the following form

$$\begin{aligned}\overline{E}^{(1)} &= C^{(2)} \circ E^{(1)} \circ A^{(1)} \circ B^{(0)} \circ \left(C^{(1)}\right)^{-1} \\ &= C^{(2)} \circ E^{(1)} \circ A^{(1)} \quad \text{since } B^{(0)} \circ \left(C^{(1)}\right)^{-1} = Id\end{aligned}$$

Moreover, note that $A^{(1)}$ is part of some affine-quadratic self-equivalence for $E^{(1)}$, meaning that the exists some quadratic function $B^{(1)}$ such that

$$B^{(1)} \circ E^{(1)} \circ A^{(1)} = E^{(1)}.$$

Thus we can write the following, remembering that $C^{(2)}$ is some invertible affine map:

$$\begin{aligned}& & \overline{E}^{(1)} &= C^{(2)} \circ E^{(1)} \circ A^{(1)} \\ \Leftrightarrow & & \left(C^{(2)}\right)^{-1} \circ \overline{E}^{(1)} &= E^{(1)} \circ A^{(1)} \\ \Leftrightarrow & & B^{(1)} \circ \left(C^{(2)}\right)^{-1} \circ \overline{E}^{(1)} &= B^{(1)} \circ E^{(1)} \circ A^{(1)} \\ \Leftrightarrow & & Q \circ \overline{E}^{(1)} &= E^{(1)}\end{aligned}$$

for some quadratic function Q.

Thus, as we can query the oracle for some input x to get $y = F(x)$ with $F = \overline{E}^{(1)}$ for the first round, we know that there is some quadratic function Q such that $z = Q(y) = E^{(1)}(x)$. The general framework of these queries is give in Figure 4.

Thus the general idea of the previous attack in the case where there is no output external encoding can be used in a similar way. One would generate a set of inputs x^i and compute the corresponding $y^i = F(x^i)$ as well as their corresponding degree 2 vectors $\tilde{y}^i$. Then we know that for the correct value of k, we could compute z^i and have the relation $z^i = M\tilde{y}^i$ for some matrix M representing the ANF of Q. However, if we naively use this approach as before, guessing bits of k one by one, we would not filter any value for k as

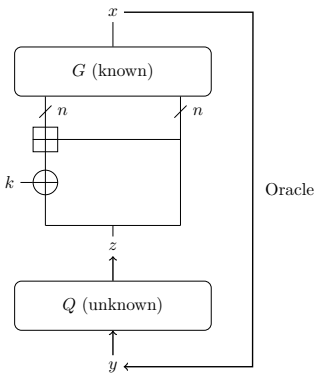

Figure 4: Structure of the function where we aim at recovering k in the case with no *input* external encoding

there is no non-linear map between the addition of the key and z. Thus, we need to be a bit smarter, and actually consider a slightly more complicated case.

Notice that if we consider the first two (encoded) rounds instead of only the first one, we actually end up with a very similar structure. Indeed, when the input external encoding is trivial, the first two encoded rounds can be written as

$$\begin{aligned}\overline{E}^{(2)} \circ \overline{E}^{(1)} &= \left[C^{(3)} \circ E^{(2)} \circ A^{(2)} \circ B^{(1)} \circ \left(C^{(2)}\right)^{-1}\right] \circ \left[C^{(2)} \circ E^{(1)} \circ A^{(1)}\right] \\ &= C^{(3)} \circ E^{(2)} \circ A^{(2)} \circ B^{(1)} \circ E^{(1)} \circ A^{(1)} \\ &= C^{(3)} \circ E^{(2)} \circ A^{(2)} \circ E^{(1)} \quad \text{since } B^{(1)} \circ E^{(1)} \circ A^{(1)} = E^{(1)}\end{aligned}$$

Thus, using the facts that $C^{(3)}$ is an invertible mapping and that we have some quadratic function $B^{(2)}$ such that

$$B^{(2)} \circ E^{(2)} \circ A^{(2)} = E^{(2)},$$

we can write

$$\begin{aligned} & \overline{E}^{(2)} \circ \overline{E}^{(1)} = C^{(3)} \circ E^{(2)} \circ A^{(2)} \circ E^{(1)} \\ \Leftrightarrow \quad & \left(C^{(3)}\right)^{-1} \circ \overline{E}^{(2)} \circ \overline{E}^{(1)} = E^{(2)} \circ A^{(2)} \circ E^{(1)} \\ \Leftrightarrow \quad & B^{(2)} \circ \left(C^{(3)}\right)^{-1} \circ \overline{E}^{(2)} \circ \overline{E}^{(1)} = B^{(2)} \circ E^{(2)} \circ A^{(2)} \circ E^{(1)} \\ \Leftrightarrow \quad & Q \circ \overline{E}^{(2)} \circ \overline{E}^{(1)} = E^{(2)} \circ E^{(1)} \end{aligned}$$

for some (unknown) quadratic function Q.

Now if we rewrite the structure of this function as before, we get what is depicted in Figure 5 (note that the key of the second round is just considered as part of the quadratic function Q).

By doing so, we now have a non-linear operation (the modular addition) between the key addition and z, which allows us to filter out wrong key guesses in a similar way as before. More precisely, we first aim at computing the LSB z_0^i of z^i for each input x^i according to some key guesses. Note that due to now having the right shift operation by α, computing this bit of z^i now requires to guess 2 bits of k (bit 0 and bit α). As before, if this key guess is correct, then we know that there must be some matrix M_0 such that

$$\begin{pmatrix} z_0^1 & z_0^2 & z_0^3 & \cdots & z_0^L \end{pmatrix} = M_0 . \begin{pmatrix} \tilde{y}^1 & \tilde{y}^2 & \tilde{y}^3 & \cdots & \tilde{y}^L \end{pmatrix},$$

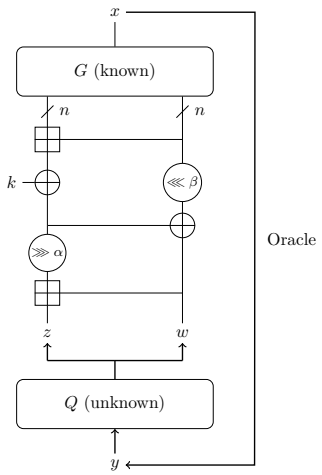

Figure 5: Structure of the function where we aim at recovering k in the case with no input external encoding when considering 2 consecutive rounds.

i.e. the vector $z_0 = (z_0^1, z_0^2, z_0^3, \ldots, z_0^L)$ must belong to the column space of the matrix $\tilde{Y}$ built with the $\tilde{y}^i$ as columns, which we can again efficiently do by precomputing a parity-check matrix for this space when generating the $\tilde{y}^i$. The whole process is then essentially the same as in the previous attack, except that we now guess bits of k two by two (until there is some overlap between previously made guesses, if any). As before, it is worth noting that in practice, after having made the j-th guess (so guessing k_j and $k_{j+\alpha}$), we are not always left with a single candidate. Unlike the previous case where we were left with only two guesses most of the time, here we are most of the time left with 16 candidates (only 4 after the first guess on k_0 and k_α obviously), which thankfully shrinks down to only 4 candidates for a given round key at the end of a given round (i.e. we get 4 candidates for the round key of the first round). Thankfully, when recovering the round key for the next round, only one among these 4 candidates allows us to get candidates for the next round key. For example, after the first round we get 4 candidates for the value of the corresponding round key $k^{(1)}$, but once we iterate the attack over the second round, we still end up with only 4 candidates overall for the round keys $(k^{(1)}, k^{(2)})$, but among these 4 candidates, the value of $k^{(1)}$ does not change (i.e. we recover the exact value of the round key $k^{(1)}$). Thus again, to recover enough key material to obtain the whole $4n$ bit master key, we only need to iterate the attack over 5 rounds, which in practice takes about 146 seconds total, including 72 seconds spent in the oracle calls. Note that the theoretical complexity of the attack is essentially the same as before, except that the constants hidden in the big-O notations are a bit larger due to needing to query one more round each time, and having to do a few more computations (both because we need to simulate 2 consecutive rounds as well as having more candidates left after each guess).

5 Tools for Analyzing the Modular Addition

In this section, we recall existing analyses of modular addition, and derive new tools which will be used in our decomposition attacks.

5.1 Differential properties of modular addition

Differential transitions through addition modulo 2^n and their probabilities were characterized by Lipmaa and Moriai [LM01].

Lemma 1 ([LM01, Lemma 3]). *The probability of a differential transition* $(\alpha, \beta) \to \gamma$ *through modular addition is nonzero if and only if*

$$\alpha_i \oplus \beta_i \oplus \gamma_i = \begin{cases} 0 & \text{if} \quad (i = 0) \ , \\ \beta_{i-1} & \text{if} \quad (i \geq 1) \wedge (\alpha_{i-1} = \beta_{i-1} = \gamma_{i-1}) \ . \end{cases} \tag{1}$$

Theorem 1 ([LM01, Algorithm 2]). *If differential transition* $(\alpha, \beta) \to \gamma$ *through modular addition has nonzero probability, then the probability is equal to* 2^{-n+l+1}*, where*

$$l = |\{i \in \{0, \ldots, n-2\} : \ \alpha_i = \beta_i = \gamma_i\}| \ . \tag{2}$$

5.2 Affine equivalence of a Boolean function to a sum of quadratic independent monomials

We consider the following problem.

Problem 1. *Let* $f : \mathbb{F}_2^n \to \mathbb{F}_2$ *be a quadratic Boolean function given by its ANF. Find a bijective linear map* $A : \mathbb{F}_2^n \to \mathbb{F}_2^n$ *and an affine map* α *such that*

$$(f \circ A)(x) = x_0 x_1 \oplus x_2 x_3 \oplus \ldots \oplus x_{2k-2} x_{2k-1} \oplus \alpha(x),$$

for a nonnegative integer k*, or show that such* A, α, k *do not exist.*

For small values of k, this problem can be solved efficiently using *linear structures*.

Definition 4 (Linear Structure). Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function. A vector $a \in \mathbb{F}_2^n$ is called a linear structure of f if $x \mapsto f(x) + f(x + a)$ is constant.

Linear structures of a Boolean function form a vector space. For a quadratic Boolean function, it can be computed as a kernel of the indicator matrix of the quadratic terms in the ANF (i.e., the matrix M such that $M_{i,j} = 1$ if and only if the ANF of $f(x)$ contains the monomial $x_i x_j$).

Let V be the vector space of linear structures of the function f from Problem 1. Then, the orthogonal complement of V, denoted $V^\perp$ is the vector space spanned the (affine images) of the variables of the monomials, that is,

$$V^\perp = \operatorname{span} \langle A_0, A_1, \ldots, A_{2k-1} \rangle.$$

For small k, recovering the correct basis $A_0, A_1, \ldots$ of the vector space $V^\perp$ can be done by an exhaustive search, testing that $f + \langle A_0, x \rangle \cdot \langle A_1, x \rangle + \ldots$ is affine. This is sufficient for the purpose of this work, as we shall only attack the problem with $k = 1$ and $k = 2$ (i.e., the case of at most two quadratic monomials). The time complexity of this method (for fixed k) is given by computing the space of linear structures and its complement, which can be done in $\mathcal{O}(n^3)$.

5.3 Equations describing the modular addition

Courtois and Debraize [CD08] studied algebraic descriptions of modular addition (for analyzing the SNOW 2.0 cipher).

Proposition 1 ([CD08, Sect.3.1,3.2]). *The n-bit addition $z = x \boxplus y$ is fully described by the following n equations:*

$$\begin{cases} z_0 = x_0 + y_0, \\ z_1 = x_1 + y_1 + x_0 y_0, \\ z_2 = x_2 + y_2 + x_1 + y_1 + x_1 y_1 + x_1 z_1 + y_1 z_1, \\ \vdots \\ z_i = x_i + y_i + x_{i-1} + y_{i-1} + x_{i-1} y_{i-1} + x_{i-1} z_{i-1} + y_{i-1} z_{i-1}, \\ \vdots \\ z_{n-1} = x_{n-1} + y_{n-1} + x_{n-2} + y_{n-2} + x_{n-2} y_{n-2} + x_{n-2} z_{n-2} + y_{n-2} z_{n-2}. \end{cases}$$

Furthermore, there are in total $6n - 2$ quadratic and 1 linear equations that can be derived from these base n equations.

We slightly reformulate these relations in terms of $Z = S(X)$, simplifying the analysis. This comes at the cost of increasing the number of equations, which is insignificant for our attacks. The following proposition characterizes all bilinear input/output relations of the bijective modular addition S.

Proposition 2. *For $8 \le n \le 64$, there exist exactly $(n^2 + 7n + 6)/2$ linearly independent Boolean polynomials $E_i(X, Z)$ in variables $X_0, \dots, X_{N-1}, Z_0, \dots, Z_{N-1}$ and of degree 1 in X and Z separately such that $E_i(X, Z) = 0$ for all i and for all $Z = S(X)$, i.e., for all $X = (x, y)$ and $Z = (x \boxplus y, y)$. These equations completely describe S.*

Proof. Experimental computation by interpolation (see Subsection 5.4). The quadratic explosion comes from the relations $y_i \tilde{y}_j = y_j \tilde{y}_i$ for $i < j$, where $X = (x, y), Z = (z, \tilde{y})$. The completeness follows from the completeness of relations from Proposition 1 and from the relations $y_i = \tilde{y}_i$, both of which have to be included in the set of relations. □

Remark 1. Compared to the quadratic relations on triples $(x, y, z = x \boxplus y)$, the bilinear relations on X, Z from the proposition are more redundant, due to the equality of the right halves of X and Z (25% more monomials and $\mathcal{O}(n^2)$ relations instead of $\mathcal{O}(n)$). However, this simplifies analysis and presentation due to the simple equation form and does not noticeably affect the complexities.

5.4 Black-box relation interpolation (affine encodings)

In this section, we describe the powerful tool of black-box relation interpolation, which will be actively used in our attacks, and has many consequences for the white-box implicit framework in general. Although we describe it in the example of bijective modular addition, we only use the fact that it has a low-degree implicit function. We first consider the case of affine encodings, which itself is a step in our attack, and later we will consider an extension of this technique to quadratic-affine encodings.

Consider the bijective modular addition S composed with (unknown) affine encodings A, B:

$$\tilde{S} = B \circ S \circ A.$$

Clearly, all input/output pairs (X, Z) of $\tilde{S}$ satisfy the relations

$$\tilde{E}_i(X, Z) \coloneqq E_i(A(X), B^{-1}(Z)) = 0$$

for all the relations E_i of S. It is easy to verify that all $\tilde{E}_i$ are also bilinear. It follows that the number and the degrees of the relations do not change under affine encodings.

Given black-box access to $\tilde{S}$, the *vector space* of these relations (i.e., some basis of it) can be reconstructed by generic interpolation. Let

$$\tilde{E}(X,Z) = \sum_{j=0}^{N-1}\sum_{k=0}^{N-1} \lambda_{j,k}^{XZ} X_j Z_k + \sum_{j=0}^{N-1} \lambda_j^X X_j + \sum_{k=0}^{N-1} \lambda_k^Z Z_k + \lambda_0, \tag{3}$$

where λ-variables denote the m coefficients, $m = N^2+2N+1$. Let $(X^{(t)}, Z^{(t)})$ be a random input/output pair of $\tilde{S}$. The equation (3) applied to this pair, that is, $\tilde{E}(X^{(t)}, Z^{(t)}) = 0$ can be viewed as a linear equation on the unknown λ-coefficients. A sufficiently large amount of random samples can be used to construct a linear system of maximum rank (with overwhelming probability). Since the rank is upper-bounded by the number m of monomials in X, Z (equal to the number of unknown λ-coefficients), the amount of $m+\epsilon$ samples (for some positive integer ϵ) would result in system of a maximum possible rank with probability $1-2^{-\epsilon}$, under natural randomization assumptions.

Remark 2. It is not strictly necessary to sample inputs uniformly at random. In particular, in the attack on quadratic encodings (Subsection 7.3), we will only use inputs with certain quadratic monomials equal to zero. While such *undersampling* may not invalidate one of the right equations $\tilde{E}(X,Z) = 0$, it may (but not necessarily) introduce extra relations, which only hold on the sampled space, which require a special care to be detected and removed from the recovered vector space of relations.

Time complexity The main step of the interpolation is solving the linear system in m variables and equations (time complexity $\mathcal{O}(m^3) = \mathcal{O}(n^6)$). The query complexity is equal to $m+\epsilon$, and the queries in the implementation of [RVP22] are effectively performed in total time $\mathcal{O}(mn^3) = \mathcal{O}(n^6)$. Here we assume that the dominating step of implicit computations is the solution of the resulting linear system (which is true if the degree of the implicit function is not very high).

5.5 Optimization and inversion attacks

In this section, we describe direct applications and extensions of black-box relation interpolation.

Inversion of addition with affine encodings The interpolated space of bilinear relations $\{E^{(i)}(X,Z) = 0\}_i$ can be viewed as an implicit function for the $\tilde{S}$ function (since these relations completely *describe* the function). Furthermore, it is quasilinear in *both directions*, allowing efficient *inversion* of the function. More precisely, the preimage X of a given output $Z = \tilde{S}(X)$ can be computed by substituting the value of Z into the equation system $\{E^{(i)}(X,Z) = 0\}_i$, and solving the resulting linear system for X. This is the same as the principle of implicit white-box implementations [RVP22].

Effectively, the black-box interpolation and implicit inversion breaks the one-wayness property of the implicit white-box ARX implementations with affine encodings. It also covers other functions with bilinear implicit functions, such as the finite field inverse function used in the AES block cipher.

Optimization of the white-box oracle The interpolated system of bilinear equations can be also used to make the white-box round oracle more efficient: even though the white-box implementation uses the same principle of implicit computation, the system of $X-Z$ relations there is much more heavier, due to high degree of polynomials in X (used to obfuscate the implicit representation), while in our case the degree in X is 1, leading to more efficient computations.

An optimized implementation of forward/inverse computation using an implicit bilinear function is described in Appendix B. For the reference, on a laptop with an Intel(R) Core(TM) i7-1185G7 3.00GHz CPU, one forward or inverse query after the precomputations takes respectively 10, 35, 100 microseconds for affine-encoded bijective modular addition on word sizes $n = 16, 32, 64$. The precomputations excluding query timings take respectively 2 seconds, 35 seconds, 7 minutes.

Extension to quadratic-affine encodings The interpolation method naturally generalizes to the case of quadratic-affine encodings. Consider the set of bilinear relations $\tilde{E}(X, Z) = 0$ satisfied by the affine-encoded bijective modular addition. Substituting X with the quadratic function $Q(W)$ of the input in the quadratic-affine case, we obtain the relations $\hat{E}(W, Z) = \tilde{E}(Q(W), Z) = 0$, where $\tilde{E}$ is bilinear. It is sufficient to extend the original bilinear monomial basis by monomials of shape $X_i X_j Z_k$, in order to cover possible monomials of these new relations $\hat{E}_i$. The time complexity thus grows to $\mathcal{O}(m^3) = \mathcal{O}(n^9)$, whereas only $\mathcal{O}(n^3)$ queries are needed.

This method can be used to *optimize* the oracle, simplifying the implicit function to a quadratic-affine one. However, this method is not directly applicable inversion of the function, because it is not quasilinear in the input variable $\hat{X}$. In general, we can not expect an efficient solution, as it effectively would perform inversion of a general quadratic map Q, which is a hard problem from the area of multivariate-quadratic (MQ) cryptography (see e.g. [KPG99]).

In practice, the quadratic encoding used in the implementation of [RVP22] is very sparse. In fact, experimentally, interpolating the protected quadratic-affine encoded in the *bilinear* basis for $n = 16$ still yielded a large number of equations (around 130 bilinear equations in the quadratic-affine case compared to 187 bilinear equations in the affine case). While these equations do not allow unique inversion, they determine the input up to 4-6 bits on average, which can be checked exhaustively using the original forward oracle. This technique thus allows quite straightforward inversion of the round oracle in the sparse-quadratic-affine case (strongest setting proposed by [RVP22]), at least for the case of $n = 16$ and for the hardcoded encoding shapes in the implementation of [RVP22]. Its generalization for larger n is yet unclear.

6 Black-box Decomposition of an ARX Round with Affine Encodings

In this section, we describe a decomposition attack against a bijective modular addition with secret *affine* encodings. The attack only requires a black-box access to the primitive, the implicit representation given in the white-box framework of [RVP22] is not required (other than to implement the oracle calls). In particular, this breaks the implicit ARX framework with affine self-equivalences (with an extra key recovery step following the decomposition of two consecutive rounds, described in Section 8). Note that a sole inversion of the rounds can be done using interpolated relations (Subsection 5.4) without any decomposition required. Furthermore, it is worth to first optimize the oracle using the optimization method from Subsection 5.5 and Appendix B, since oracle calls take a large fraction of the attack's time. The attack then can be performed "offline", that is, without calls to the actual white-box oracle, which can in principle be unnecessarily slow.

In the following, we consider the map $\tilde{S} = B \circ S \circ A$, which is the bijective modular addition with affine encodings A, B (see Figure 6).

Iterative decomposition process The decomposition process consists of many iterations of finding parts of the outward affine maps. The recovered parts can be abstracted away in

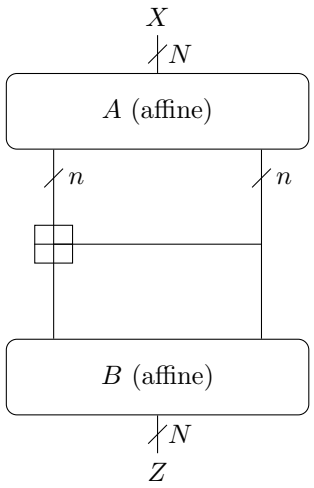

Figure 6: Bijective modular addition $\tilde{S} = B \circ S \circ A$ with encodings $A, B : \mathbb{F}_2^N \to \mathbb{F}_2^N$.

the following way. We use $\mathbb{O} : \mathbb{F}_2^N \to \mathbb{F}_2^N$ to denote the oracle function in the current step. Initially, we start with $\mathbb{O} = \tilde{S}$ implementing an affine-encoded bijective modular addition. Assume that we found affine "projection" maps π_X, π_Z such that for all $Z = \mathbb{O}(X)$, we have $\pi_X(X)$ and $\pi_Z(Z)$ agree on n least significant bits. Then, we can set new target oracle

$$\mathbb{O}' = \pi_Z \circ \mathbb{O} \circ \pi_X^{-1}.$$

It is such that $\mathbb{O}'(x, z) = (y, z)$, i.e., it captures the property of the discovered projection maps. Now, it is sufficient to decompose $\mathbb{O}'$, which should be easier as it is more structured. Indeed, if $\mathbb{O}' = B' \circ S \circ A'$, then the original $\mathbb{O}$ can be decomposed as

$$\mathbb{O} = \pi_Z^{-1} \circ \mathbb{O}' \circ \pi_X = (\pi_Z^{-1} \circ B') \circ S \circ (A' \circ \pi_X).$$

Note that the nesting decompositions does not result in a query slowdown, as we can always store a single layer of the accumulated projections maps, i.e., projection maps relative to the original oracle and not to the previous oracle.

High-level overview of the attack The decomposition procedure consists of several stages.

1. Location and alignment of the $n + 1$ linear bits in the output and their subspace in the input (the full right branch of the addition and the least significant bit of the left branch).
2. Bit-by-bit triangularization of the affine encodings using differential rank-based procedure, inspired by a similar approach from [DFLM18] applied to the ASA structure (affine-encoded layer of S-boxes).
3. Recovery of the outer triangular linear maps using differential properties of modular addition.
4. Recovery of the three Feistel affine maps (affine maps in the two outer Feistel XORs of the right branch to the left branch, and an affine map applied to the right summand), based on a bit-by-bit search of images of the maps under a fixed right-hand side and combination of linearly independent images.
5. Correction of the first carry bit expression.

The main procedure returns one of the possible decompositions; any other decomposition can be obtained by applying a certain affine self-equivalence of the bijective modular addition. A symbolic expression of such self-equivalences allows to combine candidate decompositions of two consecutive rounds and align them to match the original cipher's linear layer, yielding information about the involved subkey between the two rounds (conditioned on sufficient diffusion of the cipher's linear map). This step will be described in Section 8.

6.1 Locating linear bits

Let $z = x \boxplus y$. The first bit of modular addition is linear: $z_0 = x_0 \oplus y_0$. In the bijective modular addition $(x, y) \mapsto (x \boxplus y, y)$, the whole right branch is given in the output. Therefore, bijective modular addition has $n+1$ output bits which are linear functions of the input bits. It is also easy to verify that the other $n-1$ output bits have degree at least 2 (more precisely, the output bit z_i has degree $i+1$ when $i \leq n-2$ and degree $n-1$ when $i = n-1$) . The secret affine encodings only change the bases of the corresponding input and output affine $(n+1)$-dimensional subspaces.

Identifying the linear outputs can be done easily by searching for linear relations on the graph of the function, that is, on the concatenated inputs and outputs. To cover possible constants addition in the affine encodings, it is sufficient to add the 1 constant to each sample. The procedure is described in more details in Algorithm 1 and the resulting oracle structure after applying the projections is illustrated in Figure 7. Its complexity is $\mathcal{O}(n^3)$ time and $\mathcal{O}(n)$ queries (here, queries dominate as one query cost is at least $\mathcal{O}(n^3)$ even after optimization).

Algorithm 1 Recovering input/output matching affine subspaces

Input: oracle $\mathbb{O}$ implementing bijective modular addition with affine encodings
Output: affine maps π_x, π_z such that $\mathbb{O}' = \pi_z \circ \mathbb{O} \circ \pi_x^{-1}$ is such that $\mathbb{O}'(x||y) = (z||y)$ for all $(x, y) \in \mathbb{F}_2^{n-1} \times \mathbb{F}_2^{n+1}$, with $z \in \mathbb{F}_2^{n-1}$

for $i \in \{0, \ldots, N + \epsilon - 1\}$ **do**
  $x^{(i)} \xleftarrow{\$} \mathbb{F}_2^N$
  $y^{(i)} \leftarrow \mathbb{O}(x^{(i)})$
  $v^{(i)} \leftarrow (x^{(i)}||y^{(i)}||1)$
end for
$V \leftarrow$ matrix with rows $\{v^{(i)}\}_i$
$B \leftarrow \text{basis}(\ker V)$ so that $V \times B = 0$
$(\pi_x, \pi_z, c_y) \leftarrow B^T$, where $\pi_x, \pi_z \in \mathbb{F}_2^{(n+1)\times N}, c_y \in \mathbb{F}_2^{(n+1)\times 1}$
return π_x, $\pi_\oplus(0^{n-1}||c_y^T)$

6.2 Triangularization of the outer affine maps (left branches)

We will now use the method from [DFLM18] to partially recover the outer affine maps on the left branch. The idea is to query a fixed random input difference on the left branch and no difference on the right branch (possible due to the previous step), and to compute the dimension of the space of the observed output differences. This relies on the following differential property of modular addition.

Proposition 3. *Let* $\Delta x = (\ldots \,||\, 0^k), \Delta y = 0^n$. *Let* Z *denote the set of all possible output differences* Δz *of* $z = x \boxplus y$. *Then,* $\operatorname{rank} Z \leq n - k$. *Furthermore, if* $\operatorname{rank} Z = n - k$, *then* $(\Delta x)_k = (\Delta z)_k = 1$, *i.e.,* Δx *and* Δz *both have shape* $(\ldots \,||\, 1 \,||\, 0^k)$.

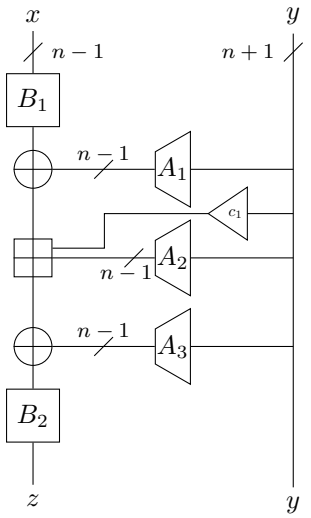

Figure 7: Remaining oracle structure after applying the recovered projections. Here, A_1, A_2, A_3, B_1, B_2 are unknown affine maps, c_1 is first carry bit of the original n-bit addition, expressible as $c_1(y) = \langle \alpha, y \rangle \cdot \langle \beta, y \rangle$ for some $\alpha, \beta \in \mathbb{F}_2^{n+1}$.

Proof. Follows from the fact that k-bit LSB zero difference in $\Delta x, \Delta y$ always leads to k-bit LSB zero difference in Δz; the first active difference bit stays active. □

Remark 3. Note that the converse is not true in general: a non-full rank does not guarantee that the k-indexed bit is zero. Furthermore, for some sparse differences, reaching full rank requires an exponential number of samples (for example, consider the difference $(0^{n-1} \,||\, 1)$, propagating to $(1 \,||\, \ldots)$ with probability $2^{-(n-2)}$).

This proposition guarantees that, on a input difference with k least significant zero bits, a full-rank space of output differences implies that the input difference and all output differences have the k-indexed bit equal to 1. This provides information about the outer affine map B_2 on the left branch. In particular, one set of differences is sufficient to recover the partial map from the $n-k$ most significant bits to the k-indexed bit in the output. In the input, since the same difference was used, the obtained information is limited. However, now we can sample $n+\epsilon$ random input differences (with k zero least significant bits on the left branch) and use the recovered linear map for the output difference to determine the k-indexed difference bit (since the first (possibly) active bit is the same in the input and in the output difference). In fact, it is not necessary to sample random input differences, it is sufficient to query each of the $n-1-k$ unit differences.

Note that at step k, we only learn the linear projection of the $n-1-k$ active most significant bits to the current least significant bit (i.e., the one indexed $n+1+k$ in the full input, or indexed k in the left $(n-1)$-bit branch). The other bits of the projection are not learnt since they are set to zero in the samples' differences. Thus, the outer linear maps on the left branch are recovered only up to a lower triangular matrix (see Figure 8). These unknown bits correspond to possible XORs from lower bits to upper bits. The missing parts are recovered in the next step.

Complexity of this step is $\mathcal{O}(n^4)$ time (solving $n-2$ linear systems) and $\mathcal{O}(n^2)$ samples (assuming the rank saturates in $\mathcal{O}(n)$ samples on average).

Algorithm 2 Triangularization of the outer maps on the left branch

Input: oracle $\mathbb{O}$ implementing bijective modular addition with partially recovered affine encodings (Subsection 6.1)

Output: affine maps π_x, π_z such that $\mathbb{O}' = \pi_z \circ \mathbb{O} \circ \pi_x^{-1}$ matches structure from Figure 8 with lower triangular T_1, T_2

Complexity: $\mathcal{O}(n^2)$ queries, $\mathcal{O}(n^4)$ time

$\mathbb{O}_0 \leftarrow \mathbb{O}$
for $k \in \{0, \dots, n-2\}$ **do**
  repeat
    $\Delta x \xleftarrow{\$} (*^{n-1-k} \,||\, 0^{n+1+k})$
    $Z \leftarrow \left\{ \Delta\mathbb{O}_k(\Delta x) \;\middle|\; n + \epsilon \text{ times} \right\}$
  until rank $Z = n - 1 - k$
  solve $Z \times \alpha = (1, \dots, 1)^T$ for $\alpha = (*^{n-1-k} \,||\, 0^{n+1+k})$
  $\pi_z \leftarrow$ a linear map such that $(\pi_z(z))_{n+1+k} = \langle \alpha, z \rangle$, $(\pi_z(z))_i = z_i$ for $i > n+1+k$
  $\beta \leftarrow \left(\pi_z(\Delta\mathbb{O}_k(e_i))_{n+1+k} \mid n + 1 + k \le i < 2n\right) \,||\, 0^{n+1+k}$
  $\pi_x \leftarrow$ a linear map such that $(\pi_x(x))_{n+1+k} = \langle \beta, x \rangle$, $(\pi_x(x))_i = x_i$ for $i > n+1+k$
  $\mathbb{O}_{k+1} \leftarrow \pi_z \circ \mathbb{O} \circ \pi_x^{-1}$
end for
return $\mathbb{O}_{n-1}$

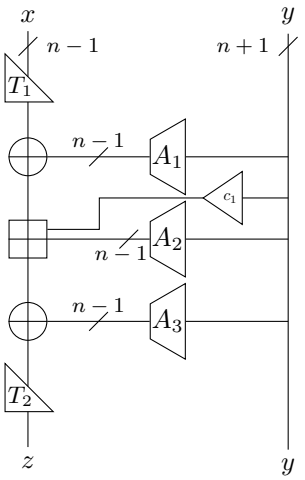

Figure 8: Remaining oracle structure after the triangularization of the outer maps on the left branches by Algorithm 2. Here, A_1, A_2, A_3, T_1, T_2 are unknown affine maps, T_1 and T_2 having linear part of a lower triangular shape, c_1 is first carry bit of the original n-bit addition, expressible as $c_1(y) = \langle \alpha, y \rangle \cdot \langle \beta, y \rangle$ for some $\alpha, \beta \in \mathbb{F}_2^{n+1}$.

6.3 Recovery of the outer affine maps (left branches)

At this step, we will recover the triangular affine maps on the left branch. The idea is to use differential properties of the modular addition.

Proposition 4. *Let $z = x \boxplus y$ be an n-bit modular addition, $n \geq 3$. Set*

$$\Delta y = 0, \quad \Delta x_1 = e_0 = (0, 0, 0, \ldots, 0, 1), \quad \Delta x_2 = e_0 \oplus e_{n-2} = (0, 1, 0, \ldots, 0, 1).$$

Then, the most probable transitions with input differences $(\Delta x_1, \Delta y)$ and $(\Delta x_2, \Delta y)$ respectively are described by

$$\Pr[(\Delta x_1, \Delta y) \xrightarrow{\boxplus} \Delta z] = \begin{cases} 1/2, & \Delta z = (0, \ldots, 0, 0, 1) = \Delta x_1, \\ 1/4, & \Delta z = (0, \ldots, 0, 1, 1), \\ \leq 1/4, & \textit{otherwise}\ldots \end{cases} \tag{4}$$

$$\Pr[(\Delta x_2, \Delta y) \xrightarrow{\boxplus} \Delta z] = \begin{cases} 1/4, & \Delta z = (0, 1, \ldots, 0, 1) = \Delta x_2, \\ 1/4, & \Delta z = (1, 1, \ldots, 0, 1) = \Delta x_2, \\ \leq 1/4, & \textit{otherwise}\ldots \end{cases} \tag{5}$$

Proof. Follows from the Lipmaa-Moriai theorem by direct computation. □

Note that Δx_1 propagates to a difference with the largest probability $1/2$, while Δx_2 propagates only with probabilities at most $1/4$. Assume that we control $n-2$ least significant bits. Then, we can distinguish Δx_1 and Δx_2 by sampling output differences and distinguishing the best probability $1/2$ from $1/4$ respectively.

Diagonals recovery In the application to the lower triangular map recovery, we consider truncated modular addition, starting from bit k and ending at bit $k + w - 1$. Thus, the value n in theorem is set to a smaller "window" w. The described technique allows to decide whether the current active least significant bit is added to the more significant bit $w - 2$ positions ahead, assuming such elementary XOR operations were already recovered for distances less than w. In this way, we recover the lower triangular map diagonal-by-diagonal, starting from the one adjacent to the main diagonal containing the largest number of unknowns (after the main diagonal which has to be all-1). The procedure is described more formally in Algorithm 3. Its complexity is $\mathcal{O}(n^3)$ time and queries. Again, queries dominate in practice with total time $\mathcal{O}(n^6)$.

Bottom row recovery Note however that this method does not recover the linear map added to the most significant bit (the bottom row of the lower triangular matrix). This is because it distinguishes $(0, 0, \ldots)$ from $(0, 1, \ldots)$, and thus requires the single 0-bit padding at the most significant bit, while recovering the penultimate significant bit.

Recall that the most significant bit is differentially linear, meaning that a single-bit difference to in this bit propagates with probability 1 to itself. Therefore, $t_\alpha \circ S \circ t_\alpha = S$, where $t_\alpha : X \mapsto X \oplus \langle \alpha, X \rangle \cdot e_{N-1}$ for some $\alpha \in \mathbb{F}_2^N, \alpha_{N-1} = 0$, i.e., t_α adds a linear function of the input to the most significant bit. Therefore, $S \circ t_\alpha = t_\alpha \circ S$, and we can set the bottom row of the input matrix T_1 arbitrarily, and recover only the bottom row of the output matrix T_2. Here, we use the following differential property of a unit difference.

Lemma 2. *Let $(\Delta e_i, 0) \xrightarrow{\boxplus} \Delta z$ be a differential transition through the n-bit addition, $i \leq n - 2$. Then, $(\Delta_z)_{i+1} = 0$ implies that $\Delta_z = e_i$.*

We can use the lemma in the following way. If we observe a differential transition $(\Delta e_i, 0) \xrightarrow{\boxplus} \Delta z$ with $(\Delta_z)_{i+1} = 0$, then the most significant bit of Δ_z will be equal to 1 if

Algorithm 3 Recovering outer lower triangular matrices

Input: oracle $\mathbb{O}$ implementing bijective modular addition with partially recovered affine encodings (Subsection 6.1, Subsection 6.2)

Output: modified oracle $\mathbb{O}'$ that matches the structure from Figure 8 with T_1, T_2 being identity matrices with extra unknown coefficients in the bottom rows only.

$\mathbb{O}_3 \leftarrow \mathbb{O}$
for $w \in \{3, \dots, n-1\}$ **do**
  $M_1 \leftarrow$ identity $(n-1) \times (n-1)$ matrix
  $M_2 \leftarrow$ identity $(n-1) \times (n-1)$ matrix
  for $i \in \{0, \dots, n-w-1\}$ **do**
    $\Delta x_1 \leftarrow e_{2n-w-i}$
    $\Delta x_2 \leftarrow e_{2n-w-i} \vee e_{2n-2-i}$
    $D_1 = \{(\Delta\mathbb{O}_w(\Delta x_1))_{2n-w-i,\dots,2n-1-i} \mid \mathcal{O}(n) \text{ times}\}$ ▷ w-bit differences
    $D_2 = \{(\Delta\mathbb{O}_w(\Delta x_2))_{2n-w-i,\dots,2n-1-i} \mid \mathcal{O}(n) \text{ times}\}$ ▷ w-bit differences
    if the most frequent difference $\Delta_Z \in D_1$ occurred $\approx 50\%$ times **then**
      $(M_1)_{n-2-i,n-w-i} \leftarrow 0$
      $(M_2)_{n-2-i,n-w-i} \leftarrow (\Delta_Z)_{w-2-i}$
    else if the most frequent difference $\Delta_Z \in D_2$ occurred $\approx 50\%$ times **then**
      $(M_1)_{n-2-i,n-w-i} \leftarrow 1$
      $(M_2)_{n-2-i,n-w-i} \leftarrow (\Delta_Z)_{w-2-i}$
    else
      retry this i
    end if
  end for
  $\mathbb{O}_{w+1} \leftarrow \mathrm{diag}(M_2, \mathrm{Id}_{n-1}) \circ \mathbb{O}_w \circ \mathrm{diag}(M_1, \mathrm{Id}_{n-1})$
  ▷ $\mathrm{diag}(M, \mathrm{Id}_{n-1})$ extends the matrix M to $N \times N$ matrix by an identity matrix
end for
return $\mathbb{O}' = \mathbb{O}_n$

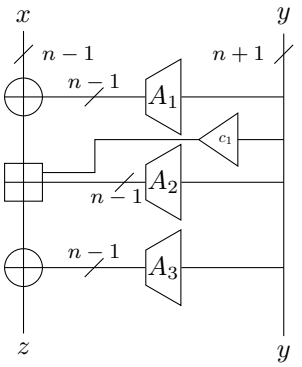

Figure 9: The structure of the oracle after recovering the outer linear maps on left branches completely. Here, A_1, A_2, A_3 are unknown affine maps, c_1 is first carry bit of the original n-bit addition, expressible as $c_1(y) = \langle \alpha, y \rangle \cdot \langle \beta, y \rangle$ for some $\alpha, \beta \in \mathbb{F}_2^{n+1}$.

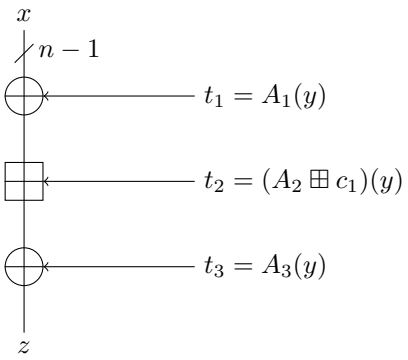

Figure 10: Structure of the oracle from Figure 9 with a fixed right branch y.

the two partial linear maps effectively add the i-th bit to the most significant bit. Since we can assume the input partial map to do nothing, we can effectively learn one bit of the bottom row of the output map. Since the method requires $i \leq n-4$ for the $(n-1)$-bit addition to be effective, one bit of the bottom row can not be recovered in this way, namely, the one defining the presence of the elementary operation $x_{n-2} \leftarrow (x_{n-2} \oplus x_{n-3})$. This bit could be guessed, but in fact, its value can be set arbitrarily, affecting only the Feistel maps recovered in the next subsections.

The remaining structure implemented by the oracle is illustrated in Figure 9.

6.4 Recovery of the Feistel affine maps (right to left XOR and addition)

The Feistel affine maps can be recovered in the following way. First, we randomly fix the right branch value y. We obtain a $(n-1)$-bit mapping $(\oplus t_3) \circ (\boxplus t_2) \circ (\oplus t_1)$ on the left branch, where $t_1, t_2, t_3 \in \mathbb{F}_2^{n-1}$ are constants depending on the fixed y (see Figure 10). The following proposition shows that there are exactly 8 solutions for t_1, t_2, t_3 for such an oracle, given that the least significant bit of t_2 is 1 (which happens for 1/2 choices of y).

Proposition 5. *Let $n \geq 2$, $t_1, t_2, t_3 \in \mathbb{F}_2^n$ with $(t_2)_0 = 1$. Then, all the maps $(\oplus t'_3) \circ$*

$(\boxplus t'_2) \circ (\oplus t'_1)$ *are equivalent to the map* $(\oplus t_3) \circ (\boxplus t_2) \circ (\oplus t_1)$ *with*

$$t'_1 = t_1 \ \oplus \ a_1 \cdot (1, 0, \ldots, 0) \ \oplus \ a_3 \cdot (1, \ldots, 1), \tag{6}$$
$$t'_2 = t_2 \ \oplus \ (a_1 \oplus a_2) \cdot (1, 0, \ldots, 0) \ \oplus \ a_3 \cdot (1, \ldots, 1, 0), \tag{7}$$
$$t'_3 = t_3 \ \oplus \ a_2 \cdot (1, 0, \ldots, 0) \ \oplus \ a_3 \cdot (1, \ldots, 1), \tag{8}$$
$$\tag{9}$$

for any $a_1, a_2, a_3 \in \mathbb{F}_2$*. In particular, there are 8 equivalent triples, whenever the least significant bit of* t_2 *is 1.*

Proof. Since the MSB of the modular addition is differentially linear, it is trivial to see that the maps $(\oplus t'_3) \circ (\boxplus t'_2) \circ (\oplus t'_1)$ are all equivalent to the map $(\oplus t_3) \circ (\boxplus t_2) \circ (\oplus t_1)$ with

$$\begin{aligned} t'_1 &= t_1 \ \oplus \ a_1 \cdot (1, 0, \ldots, 0), \\ t'_2 &= t_2 \ \oplus \ (a_1 \oplus a_2) \cdot (1, 0, \ldots, 0), \\ t'_3 &= t_3 \ \oplus \ a_2 \cdot (1, 0, \ldots, 0), \end{aligned}$$

for all $a_1, a_2 \in \mathbb{F}_2$. Thus, we only need to focus on the case where $a_1 = a_2 = 0$ and $a_3 = 1$. For some $x \in \mathbb{F}_2$, denote $z = ((\oplus t_3) \circ (\boxplus t_2) \circ (\oplus t_1))(x)$, i.e. we have

$$z = [(x \oplus t_1) \boxplus t_2] \oplus t_3 \Leftrightarrow (z \oplus t_3) = (x \oplus t_1) \boxplus t_2.$$

Using the fact that

$$x \oplus (1, \ldots, 1) = \neg x = \boxminus x \boxminus 1 \quad \text{and} \quad \neg(x \oplus y) = x \oplus \neg y,$$

we can deduce

$$\begin{aligned} z \oplus \neg t_3 = \neg(z \oplus t_3) &= \boxminus(z \oplus t_3) \boxminus 1 \\ &= \boxminus(x \oplus t_1) \boxminus t_2 \boxminus 1 \\ &= \neg(x \oplus t_1) \boxplus 1 \boxplus \neg t_2 \boxplus 1 \boxminus 1 \\ &= (x \oplus \neg t_1) \boxplus (\neg t_2 \boxplus 1) \end{aligned}$$

Since the LSB of t_2 is 1, the LSB of $\neg t_2$ is 0 and thus

$$(\neg t_2 \boxplus 1) = \neg t_2 \oplus 1 = t_2 \oplus (1, \ldots, 1, 0).$$

In the end, on one hand we have $z = [(x \oplus t_1) \boxplus t_2] \oplus t_3$, on the other

$$\begin{aligned} z &= [(x \oplus \neg t_1) \boxplus (t_2 \oplus (1, \ldots, 1, 0))] \oplus \neg t_3 \\ &= [(x \oplus t_1 \oplus (1, \ldots, 1)) \boxplus (t_2 \oplus (1, \ldots, 1, 0))] \oplus (t_3 \oplus (1, \ldots, 1)) \\ &= [(x \oplus t'_1) \boxplus t'_2] \oplus t'_3 \end{aligned}$$

Hence $[(x \oplus t_1) \boxplus t_2] \oplus t_3 = [(x \oplus t'_1) \boxplus t'_2] \oplus t'_3$ meaning that the two mappings are equal. □

Since the proposition works for any $n \geq 2$, it follows that there are 8 solutions for each of the least significant truncations of the function. Therefore, the solutions can be efficiently recovered in a bit-by-bit search (modulo 2, modulo 4, modulo 8, ..., modulo 2^{n-1}). We observed experimentally that there are at most 8 solutions at any step. In case the number of solutions is more than 8 (at any intermediate step), we can choose another constant for y. The cost of this step is $\mathcal{O}(n^2)$, assuming $\mathcal{O}(n)$ random samples (data) are sufficient to discard all false-positives. In principle, one can craft a guaranteed number of $\mathcal{O}(n)$ samples by ensuring all possible combinations of bits and carry bits *locally* at the current guessing position. In practice, a slightly larger number of random samples is sufficient for the goal.

Combining solutions After collecting solution groups for $n+2$ independent values of y (i.e., forming a basis for $(n+1)$-dimensional affine map), we can combine them into the candidates for the maps $A_1, (A_2 \boxplus c_1), A_3$. Note that there are 8^{n+1} candidates in total. However, we observed experimentally that almost any candidate leads to a correct decomposition with high probability. Therefore, we select one of the 8 solutions in each group arbitrarily, and obtain the maps A_1, A_3 and samples of $A_2 \boxplus c_1$, which define the map A_2 up to addition/subtraction of 1 (depending on whether $c_1(y) = 0$ or $c_1(y) = 1$, which we can not compute yet). We denote by B the affine map that agrees with $A_2 \boxplus c_1$ on the used samples.

Complexity of collecting and combining the $n+2$ solutions is $\mathcal{O}(n^3)$ time and $\mathcal{O}(n^2)$ data.

6.5 Correction of the first carry bit

After removing the linear Feistel maps A_1, A_3 via projections, we are left with the map

$$\mathbb{O} : \mathbb{F}_2^{n-1} \times \mathbb{F}_2^{n+1} \to \mathbb{F}_2^{n-1} \times \mathbb{F}_2^{n+1} : (x, y) \mapsto (x \boxplus B(y) \boxplus c'(y), y)$$

where $B : \mathbb{F}_2^{n+1} \to \mathbb{F}_2^{n-1}$ is an affine map and $c'(y) \in \{-1, 0, 1\}$ is what we shall call a *pseudocarry*. We know that this mapping should be affine-equivalent to

$$(x, y) \mapsto (x \boxplus y_{n+1\ldots,2} \boxplus y_0 y_1, y),$$

where $y_0 y_1$ is the usual carry function (recall that y contains the two least significant bits of both operands of the target n-bit addition).

If B is not full-rank, we can randomize it by choosing another solutions for $A_1, (A_2 \boxplus c_1), A_3$ in the previous step. Otherwise, we complete it arbitrarily to an invertible affine map $B' : \mathbb{F}_2^{n+1} \to \mathbb{F}_2^{n+1}$. Then, using the projections

$$\pi_x : (x, y) \mapsto (x, B'(y)), \tag{10}$$

$$\pi_y : (x, y) \mapsto (x, B'^{-1}(y)), \tag{11}$$

we obtain a new oracle $\mathbb{O}' = \pi_y \circ \mathbb{O} \circ \pi_x^{-1}$ such that

$$\mathbb{O}' : (x, y) \mapsto (x \boxplus y_{n+1,\ldots,2} \boxplus c'(B^{-1}(y))).$$

The next step is to correct the pseudocarry sign. We observed that the preimages of -1 under $c' \circ B^{-1}$ are described by an affine map $\tau : \mathbb{F}_2^{n+1} \to \mathbb{F}_2$, that is, $\tau(y) = 1$ if and only if $c'(B^{-1}(y)) = -1$. Thus, we can correct the sign by adding the function τ to the least significant bit of y (and cancelling it in the output). If such map happens to be non-invertible, we can re-randomize the state in the previous step. As a result, we obtain an oracle $\mathbb{O}''$ mapping

$$(x, y) \mapsto (x \boxplus y_{n+1,\ldots,2} \boxplus c''(y),$$

where $c''(y) \in \{0, 1\}$ is a quadratic map. Such c'' has to be extended affine-equivalent to a quadratic monomial, i.e., have shape $c''(y) = \alpha(y) \cdot \beta(y) \oplus \gamma(y)$. Using the linear structures method Subsection 5.2, we can find affine maps α, β, γ decomposing c''.

Removing the affine part Note that $\alpha(y) \cdot \beta(y)$ matches the expected shape $y_0 \cdot y_1$, but the addition of $\gamma(y)$ breaks this structure. In order to remove γ, we use the following property of the addition.

Proposition 6. *Let $z = x \boxplus y$ be an n-bit addition. Then, $\neg z = \neg x \boxplus \neg y \boxplus 1$.*

Proof. Follows from the "two's complement": $\neg z = -z - 1$. □

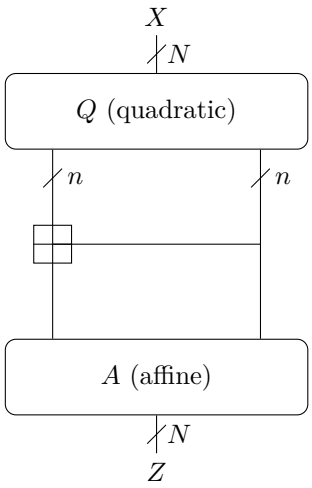

Figure 11: Bijective modular addition $\mathbb{O} = A \circ S \circ Q$ with quadratic input encoding Q and affine output encoding A, $Q : \mathbb{F}_2^N \to \mathbb{F}_2^N$.

Note that bitwise complement $\neg$ can be replaced by addition of $f(y) \cdot (1, \ldots, 1)$ to all inputs and outputs, where $f(y)$ is an arbitrary function of y. This requires that $y \oplus f(y) \cdot (1, \ldots, 1)$ is an invertible map. In a way, this applies the previous proposition selectively, only on inputs with $f(y) = 1$.

Proposition 7. *Consider the setting from the text above. Let $\sigma(x, y)$ be an affine map such that*

$$\sigma_i(x, y) = \begin{cases} \beta(y), & i = 0, \\ y_{2+i-1} \oplus \gamma(y), & 1 \le i \le n-1, \\ \alpha(y) \oplus \beta(y), & i = n, \\ x_i \oplus \gamma(y), & n+1 < i < N. \end{cases}$$

If it is invertible, then, $\sigma \circ \mathbb{O}'' \circ \sigma^{-1}$ implements $(x, y) \mapsto (x \boxplus y, y)$ for $x, y \in \mathbb{F}_2^n$.

Using this proposition, we can finally obtain the bijective modular addition, and the final decomposition can be reconstructed by composing all the used projection maps. Similarly to previous steps, if the map is not invertible, we can randomize choices done in the previous steps and repeat.

The total complexity of the first carry bit correction is dominated by linear algebra and linear structure computation steps, taking $\mathcal{O}(n^3)$ time, and $\mathcal{O}(n^2)$ samples for verifications and ANF computations.

Total decomposition complexity The final complexity of decomposing an affine-encoded bijective modular addition is dominated by the steps described in Subsection 6.2 and Subsection 6.3 (triangularization and recovery of triangle maps) due to $\mathcal{O}(n^3)$ queries used in both. The former one has dominating pure computational time of $\mathcal{O}(n^4)$.

7 Black-box Decomposition of an ARX Round with (Sparse) Quadratic-Affine Encodings

We now move on to the main cryptanalysis target - bijective modular addition with quadratic-affine encodings as suggested by [RVP22]. Again, we are going to only use the

white-box round oracle, and ignore its implementation. Therefore, our attack is independent of graph automorphisms, which are used to obfuscate the implicit implementation of rounds.

Observations on encodings generated by the method of [RVP22] The quadratic input encoding of the round can not be chosen arbitrarily (except the external input encoding). It has to come from an affine-quadratic self-equivalence. More precisely, the quadratic input encoding is the quadratic part of the self-equivalence, pre-composed with a random affine map, and followed by the cipher's linear layer. Although the authors of [RVP22] found a large number of possible affine-quadratic self-equivalences, the structure of quadratic parts of them is very restricted.

Proposition 8. *Let $A, Q : \mathbb{F}_2^N \to \mathbb{F}_2^N$ be an affine-quadratic self-equivalence of bijective modular addition S and assume $n \geq 4$. Then, there exist at least n linearly independent linear combinations of outputs of Q that have degree 1.*

Proof. We have $Q \circ S \circ A = S$. Let $z = x \boxplus y$, $Z = X \boxplus Y$ correspond respectively to inner and outer variables, such that $(x, y) = A(X, Y)$ and $(Z, Y) = Q(z, y)$. Let A_2 be a part of A^{-1} such that

$$(X_0||Y) = A_2(x, y) = A_2(z \boxminus y, y).$$

Furthermore, let Q_2 be a part of Q such that

$$(Z_0||Y) = Q_2(z, y).$$

Recall that the LSB Z_0 is equal to $X_0 \oplus Y_0$. Denote by B the affine map $(X_0||Y) \mapsto (Z_0||Y)$. We have

$$B(A_2(z \boxminus y, y)) = (Z_0||Y) = Q_2(z, y).$$

That is, the part $Q_2(z, y)$ of the quadratic encoding Q can be computed by an affine map applied to $(z \boxminus y, y)$. Note that $z \boxminus y$ has exactly one quadratic output bit. It follows that Q_2 has at most 1 quadratic output bit, and the other n bits are linear (note that Q_2 outputs $n + 1$ bits). □

This proposition proves that there *all* affine-quadratic self-equivalences of bijective modular addition have at least n linear outputs, meaning that only at most half of the output of Q can be quadratic (up to affine equivalence). In practice, we studied the encodings generated by the method of [RVP22] (which are chosen in a very restricted shape). We observed that the outputs of Q have *at most 3 independent quadratic functions*. Furthermore, up to affine equivalence, these functions consist of only 1 or 2 quadratic monomials (i.e., we count a function $\langle \alpha, x \rangle \cdot \langle \beta, x \rangle$ as one monomial). While the variability of Q is very high, as claimed by the authors, this is only due to the variability of the linear combinations in the involved monomials. We conclude that the quadratic self-equivalences of bijective modular addition have very sparse quadratic part and set to exploit this weakness. Note that already the decomposition of the affine-affine encoded bijective modular addition is a nontrivial task (as can be seen from Section 6), and adding just a few (unknown) quadratic monomials significantly adds up to the complexity of the decomposition process.

High-level overview of the attack The procedure consists of several steps. The main high-level idea is to catch the unknown quadratic monomials among the quadratic outputs of the bijective modular addition (which appear due to the cipher's linear map mixing the outputs of Q to the right branch, which is present among linear combinations of output bits), and use algebraic relations from Subsection 5.4 to recover Q. The overall plan is thus as follows.

1. Locate quadratic output bits (up to addition of linear output bits), by using the zero-sum property over 3-dimensional spaces.
2. Decompose each possible linear combination of quadratic output bits into a quadratic monomial or a sum of two monomials (up to addition of linear terms), when possible.
3. Using inputs to the recovered monomials as the basis for the quadratic terms in the quadratic input encoding (this is an assumption based on the used shape of encodings), perform black-box interpolation of affine-encoded bijective modular addition on a subset of samples not triggering any quadratic monomials.
4. A composition of the encryption oracle with the system-based inversion results in the output of the quadratic encoding (because the system captures the affine-encoded addition only). Thus, the quadratic encoding can be easily evaluated and reconstructed.
5. The quadratic encoding can be inverted by computing a Gröbner basis in lexicographic monomial order. (In general, inversion of quadratic maps is a hard problem, so we can't hope for a generic robust solution.) Due to sparseness, the Gröbner basis method works very well in practice. Note that the inverse of the quadratic encoding does not have to be quadratic, it can have much higher degree.
6. As a result, we can invert the quadratic encoding on any input, and pass it to the original oracle, effectively stripping the quadratic encoding, and leaving the affine-encoded bijective modular addition, which can be decomposed as described in Section 6.

7.1 Locating quadratic output bits

The first step is to recover the linear combinations of output bits that are quadratic functions of the input bits. This can be done efficiently using the fact that any quadratic function sums to zero over any 3-dimensional affine subspace. To exploit it, let us compute multiple such sums over random 3-dimensional subspaces of the oracle function. Each such sum is an N-bit vector and the zero-sum linear combinations of these vectors form the vector space of output bits that have degree at most 2. Since we can recover the linear bits using the method from Subsection 6.1, we are interested in output bits of degree precisely 2. To filter out the output linear bits, we can simply remove them from the output. In this way, we can not lose a possible quadratic output, since adding a linear function can not change the degree of a quadratic function. The procedure is summarised in Algorithm 4. Its time complexity is $\mathcal{O}(n^3)$ and data complexity is $\mathcal{O}(n)$.

Algorithm 4 Recovering output quadratic bits

Input: oracle $\mathbb{O}$ implementing bijective modular addition with quadratic-affine encodings, with output linear bits removed

Output: linear map $\pi_q : \mathbb{F}_2^N \to \mathbb{F}_2^t$ such that $\pi_q \circ \mathbb{O}$ has all its output linear combinations having degree 2

1: **for** $i \in \{0, \dots, N + \epsilon = 1\}$ **do**
2: $\quad X \xleftarrow{\$}$ random 3-dimensional affine subspace of $\mathbb{F}_2^N$
3: $\quad v^{(i)} \leftarrow \bigoplus_{x \in X} \mathbb{O}(x)$
4: **end for**
5: $V \leftarrow$ matrix with rows $\{v^{(i)}\}_i$
6: $B \leftarrow \text{basis}(\ker V)$ so that $V \times B = 0$
7: **return** $\pi_q : x \mapsto B \times x$

7.2 Decomposition of quadratic outputs into 1 or 2 quadratic monomials

The ANFs of the recovered quadratic bits can be recovered in $\mathcal{O}(N^2)$ queries and time by using the fact that the coefficient $\lambda_{i,j}$ of the monomial $x_i x_j$ in a Boolean function f is given by $f(e_i \oplus e_j) \oplus f(e_i) \oplus f(e_j) \oplus f(0)$. Alternatively, the same amount of random queries can be used (e.g., from a common data pool shared between steps), with a generic linear algebraic interpolation step costing $\mathcal{O}(N^6)$ time.

After the first step, based on experimental data, we obtain 2-4 quadratic output bits (i.e., $\pi_q \circ \mathbb{O}$ is a quadratic map $\mathbb{F}_2^N \to \mathbb{F}_2^t$ with $2 \leq t \leq 4$), independently of the word size n. Compared on the previous observation that Q has at most 3 independent quadratic outputs, we may obtain an extra quadratic output which is the second least significant bit $z_1 = x_1 + y_1 + x_0 y_0$. The second step consists in finding affine equivalent representation of the map $\pi_q \circ \mathbb{O}$ consisting of quadratic functions with 1 or 2 quadratic monomials each. That is, we search for affine maps B, A such that each output bit of $B \circ \pi_q \circ \mathbb{O} \circ A$ contains at most 2 quadratic monomials. Note that the restriction to 2 monomials is artificial (based on the currently observed encodings) and a potentially new encoding with a few more monomials can still be attacked with the same method.

The motivation for this step comes from experimental observation of used quadratic encodings in the implementation of [RVP22], which only have a few distinct quadratic monomials (up to affine equivalence). The goal is thus to catch these quadratic monomials in the output bits of the oracle. This is in particular possible due to the cipher's linear layer mixing these quadratic monomials into the linear right branch of the modular addition, which in turn leaks the monomials in the output of the oracle.

The step's goal can be straightforwardly achieved by enumerating all linear combinations (at most 15 if $t \leq 4$) of $\pi_q \circ \mathbb{O}$ and attempting to apply the monomial decomposition method based on linear structures from Subsection 5.2. Then, we choose an arbitrary linearly independent subset of the successfully decomposed linear combinations which defines the output linear map B described above; the input linear map A maps the space of linear combinations of input bits involved in the quadratic monomials to single input bits. For example, if monomials $(x_1 + x_2 + x_5)(x_3 + x_4)$, $(x_3 + x_4, x_7)$, $(x_1 + x_2 + x_5 + x_7, x_8)$ are chosen, we map $(x_1 + x_2 + x_5), (x_3 + x_4), (x_7), (x_8)$ to four distinct input bits (we use the fact that $x_1 + x_2 + x_5 + x_7$ is expressible as a sum of new bits).

The time complexity of this step is dominated by computing the $2^t - 1$ linear structure spaces, which is done in $\mathcal{O}(2^t n^3)$ time (here, enumeration of candidates of 2 monomials among 4 linear functions takes an extra but constant time factor). The ANF computation requires $\mathcal{O}(n^2)$ queries.

7.3 Algebraic recovery of a sparse quadratic encoding

The next goal is to recover the quadratic encoding, more precisely, its quadratic part. Since the monomials themselves were recovered in the previous step, it is only missing to know where each monomial is added.

We recall the setting of Subsection 5.4. Let $Z = (B \circ S)(X)$ for some affine map B. Then, X and Z satisfy a set of bilinear relations

$$\tilde{E}(X, Z) = \sum_{j=0}^{N-1} \sum_{k=0}^{N-1} \lambda_{j,k}^{XZ} X_j Z_k + \sum_{j=0}^{N-1} \lambda_j^X X_j + \sum_{k=0}^{N-1} \lambda_k^Z Z_k + \lambda_0. \tag{12}$$

Now, let $X = Q(W)$, where Q is the quadratic bijective encoding, so that $Z = \mathbb{O}(W) = (B \circ S \circ Q)(W)$. We assume that Q can be expressed as $Q(W) = W + q(W)$ up to a linear map applied to W, where q is a purely quadratic function, i.e., all its output bits contain

only quadratic monomials, with the ANFs

$$q_j(W) = \sum_{(a<b)\in I^2} \alpha^j_{a,b} W_a W_b.$$

where I is the set of indexes of input bits involved in the quadratic monomials of the output quadratic bits (recovered in the previous step).

Remark 4. While this assumption does not often hold directly, Q may be partially "randomized" by composing the oracle with a random constant addition, which leads to generation of new linear terms from quadratic ones (for example, $(W_0{+}1)W_1 = W_0W_1{+}W_1$). We found this sufficient for our attacks. Alternatively, we could add a few extra linear terms/variables to the hypothesized expression of Q to ensure invertability of the linear part with high probability.

Substituting X by $Q(W)$ into Equation 12 gives new relations $\hat{E}$ on W, Z:

$$\hat{E}(X, Z) = \tilde{E}(Q(W), Z) = \tilde{E}(W + q(W), Z) \tag{13}$$

$$= \sum_{j=0}^{N-1}\sum_{k=0}^{N-1} \lambda^{XZ}_{j,k}(W_j + q_j(W))Z_k + \sum_{j=0}^{N-1} \lambda^X_j(W_j + q_j(W)) + \sum_{k=0}^{N-1} \lambda^Z_k Z_k + \lambda_0 \tag{14}$$

$$= \sum_{(a<b)\in I^2}\sum_{k=0}^{N-1} \hat{\lambda}^{WWZ}_{a,b,k} W_a W_b Z_k + \sum_{(a<b)\in I^2} \hat{\lambda}^{WW}_{a,b} W_a W_b \tag{15}$$

$$+ \sum_{j=0}^{N-1}\sum_{k=0}^{N-1} \lambda^{XZ}_{j,k} W_j Z_k + \sum_{j=0}^{N-1} \lambda^X_j W_j + \sum_{k=0}^{N-1} \lambda^Z_k Z_k + \lambda_0. \tag{16}$$

The new coefficients $\hat{\lambda}^{WWZ}_{a,b,k}, \hat{\lambda}^{WW}_{a,b}$ of monomials $W_aW_bZ_k, W_aW_b$ are functions of the coefficients $\lambda^{XZ}_{j,k}, \lambda^X_j$ and the coefficients $\alpha^j_{a,b}$ of the coordinate functions q_j of q:

$$\hat{\lambda}^{WWZ}_{a,b,k} = \sum_{j=0}^{N-1} \lambda^{XZ}_{j,k} \cdot \alpha^j_{a,b}, \qquad \hat{\lambda}^{WW}_{a,b} = \sum_{j=0}^{N-1} \lambda^X_j \cdot \alpha^j_{a,b}. \tag{17}$$

Essentially, the monomial basis for the relations is expanded by the terms $W_aW_bZ_k$ and W_aW_b, the number of which is $\mathcal{O}(|I|^2N)$, which can be much less compared to N^3 from the general quadratic encoding case. Experimentally, the size of I is at most 7 for the quadratic encodings used in [RVP22].

These relations can be directly interpolated in a black-box way as described in Subsection 5.4. Note that black-box interpolation only recovers the full vector space of all relations, so that we can only obtain linear combinations of all relations. It is easy to see that any linear combination of relations $\tilde{E}^{(i)}$ still follows the same structure.

However, another problem is that *other* relations may appear, not following the structure described above. They may appear due to expanded monomial basis and presence of the quadratic encoding interacting with the modular addition. There can be at most $\binom{|I|}{2}$ extra such relations (21 if $|I| \le 7$), depending on the quadratic encoding used, in the implementation of [RVP22].

We resolve this problem in the following way.

1. First, we interpolate relations in the bilinear basis $(1, W, Z, WZ)$, but using only samples satisfying $W_aW_b = 0$ for all pairs $(a, b) \in I^2$. On these samples, we have $\tilde{E}(W, Z) = \hat{E}(W, Z)$ since the terms quadratic in W are equal to zero. As a result, we directly obtain some relations of the bijective modular addition without the quadratic part of the quadratic encoding, plus possibly a few extra relations (in the

same basis). Note that the relations are all linearly mixed together as we can only interpolate the vector space that they span. The time and data complexities of this step are respectively $\mathcal{O}(n^6)$ and $\mathcal{O}(n^2)$.

2. Second, we aim to remove the extra relations from the system, in order to obtain a pure system of relations of the bijective modular addition (up to affine encodings). The idea is to choose random input/output pairs $(W^{(t)}, Z^{(t)})$, for which the interpolation-based inversion fails to determine a unique preimage (if this does not happen, it must be that we invert the bijective modular addition, leading to recovery of Q). Furthermore, we require that all inputs $W^{(t)}$ share the bits indexed by I. Then, the "good" relations can be fixed by replacing $W^{(t)}$ with $W^{(t)} + q(W^{(t)})$, which is a linear function on the coefficients $\alpha^j_{a,b}$ since $W^{(t)}$ is known. Among the "bad" relations, on the other hand, there must exist at least one that is not satisfied by the correct solution (otherwise, the inversion must have succeeded). We introduce an "error" variable $\epsilon^{(i)}$ per each relation $\hat{E}^{(i)}$ in the system, leading to linear equations of the form
$$\hat{E}^{(i)}\left(W^{(t)} + q(W^{(t)}), Z^{(t)}\right) + \epsilon^{(i)} = 0,$$
which are equations on the coefficients of q and all error variables $\epsilon^{(i)}$ (since W, Z are known). By solving this linear system with a sufficient number of samples, we find which relations are erroneous on this input/output set. Then, we can add one of the erroneous relations to all other erroneous relations to cancel the error (i.e., canceling the original mixed-in erroneous relations). This step removes one unwanted relation from the system. After repeating this step a sufficient number of times (i.e., the number of extra relations), we obtain a clear system of relations for the affinely encoded bijective modular addition $B \circ S$.

   Assuming constant number of erroneous relations, the time complexity is $\mathcal{O}(n^6)$ (solving the linear system of size $\mathcal{O}(n^2)$ relations), while the data complexity is negligible.

3. Finally, we can use the system to invert the bijective modular addition part from the oracle (using its bilinearity, as in Subsection 5.5), which allows to compute $X = Q(W)$ as $X = ((B \circ S)^{-1} \circ \mathbb{O})(W) = ((B \circ S)^{-1} \circ (B \circ S \circ Q))(W) = Q(W)$. Then, we can obtain the ANF of Q in $\mathcal{O}(n^2)$ such queries (each costs $\mathcal{O}(n^3)$) using standard methods. This costs $\mathcal{O}(n^5)$ time in total for this step.

We remark that the system of linear equations from step may have more than 1 solution due to possible quadratic-affine self-equivalences of the addition. In practice, we observe 2 possible solutions per added equation, which produces a small amount of candidates to test.

7.4 Inversion of quadratic encoding

In general, the problem of inverting a function given by quadratic polynomials is considered to be hard. In particular, multivariate quadratic (MQ) cryptography relies on hardness of this problem (a prominent example is the unbalanced oil and vinegar scheme [KPG99]). If a white-box implementation uses a generic quadratic function as an external input encoding, it can not be easily inverted. Thus, a white-box designer in principle may use an MQ public-key encryption scheme as an external encoding and it won't be breakable or invertible. This would be of course not a white-box achievement. This discussion can be seen as an argument against external encodings being a reasonable assumption.

From the general hardness it follows that we have to use a particular shape of the quadratic function if we want to invert it. The case of quadratic encodings from [RVP22] is very sparse (only a few distinct quadratic monomials are used. In most cases, an inverse

function can be computed by hands. Yet, due to several different shapes of the encoding, automating the ad-hoc process is not a straightforward and clean solution. To that end, we resort to use a generic algebraic method based on Gröbner basis (for detailed description, we refer e.g. to [CLO10]). It performs very well on sparse encodings, removing the need of manual work, and at the same time, this approach is the best known way to approach inversion of a generic quadratic encoding. Thus, it covers both the sparse encodings used in the white-box implementation that we attack and potential dense encodings in the case described in the previous paragraph.

The first method is to encode the inversion problem for each target output $y = Q(x)$, where Q is the quadratic function. This is done by considering the ideal

$$I = \langle y_0 - Q_0(x),\ \ldots,\ y_{N-1} - Q_{N-1}(x) \rangle \tag{18}$$

of the polynomial ring $\mathbb{F}_2[x_0, \ldots, x_{N-1}]$ (the target output y is a known constant). The goal is to compute the associated variety (i.e., the solution set), which can be done through Gröbner basis computations. If Q is bijective (as should be in the white-box setting), the variety should contains the only solution corresponding to $y = Q(x)$.

The second method is to compute the polynomial representation of the inverse function Q^{-1} (here, we allow the polynomial of y_i to also use variables y_j for all $j > i$). The advantage is that it requires to compute a Gröbner basis only once for a given function Q, and inversion of a given output is done simply by evaluating the computed polynomials. The disadvantage of the method is that the polynomial representation of Q^{-1} may be not compact, especially in the case of dense Q. To achieve the goal, we use an ideal I' of the same shape as the ideal I from (18), but over the polynomial ring $\mathbb{F}_2[x_0, \ldots, x_{N-1}, y_0, \ldots, y_{N-1}]$. That is, we consider x to be a vector of variables. Then, we compute the Gröbner basis of this ideal in lexicographic order (using $x_0 > \ldots > x_{N-1} > y_0 > \ldots y_{N-1}$). If Q is invertible, this results in the sequence of polynomials

$$x_{N-1} - f_{N-1}(y_0, \ldots, y_{N-1}), \tag{19}$$
$$x_{N-2} - f_{N-2}(x_{N-1}, y_0, \ldots, y_{N-1}), \tag{20}$$
$$\vdots \tag{21}$$
$$x_0 - f_0(x_1, \ldots, x_{N-1}, y_0, \ldots, y_{N-1}), \tag{22}$$

yielding the desired polynomials f_i to compute $x = Q^{-1}(y)$.

When attacking the implementation of [RVP22], we used the second method which showed to be very efficient for the sparse quadratic encodings used. It takes negligible time compared to other steps of the attack and requires no manual work. We conclude that the time complexity of the black-box decomposition attack is dominated by the algebraic recovery of the quadratic encoding Q, which requires $\mathcal{O}(n^6)$ time and $\mathcal{O}(n^2)$ queries.

This concludes the decomposition and inversion attack on the implicit ARX-based white-box scheme of [RVP22]. In the next section, we show how to use the obtained decomposition and inversion method to fully recover (most of) round subkeys and, ultimately, the master key of the underlying Speck cipher.

8 Combining Round Decompositions for Full Key Recovery in the case of Speck Cipher

In this section, we describe how to chain several consecutive round decompositions and extract the round subkeys in-between, allowing full master key recovery in the case of white-box Speck cipher implementation. More precisely, for a Speck version with W master key words, it is sufficient to chain $W + 1$ consecutive round decompositions to obtain at

most 2^{2W} master key candidates. An extra decomposed round can be used to narrow down this small set of candidates to the unique correct one. This attack relies on properties of the Speck linear layer, sparsity of affine self-equivalences of bijective modular addition, and the key schedule of Speck. This method is an extension of the technique from [VRP22] to include quadratic self-equivalences.

Remark 5. In practice, two consecutive round decompositions narrow down possible encodings to only a few candidates, which allow to decompose all consequent rounds in a much simpler way, since one side of encoding is given from the previous round.

The initial setting for this section is as follows. We are given a chain of maps $F^{(1)}, F^{(2)}, \ldots, F^{(r)}$, each being either a bijective modular addition, an affine map, or a quadratic map, such that $F^{(r)} \circ \ldots \circ F^{(1)}$ is affine-equivalent to an r'-round Speck:

$$F^{(r)} \circ \ldots \circ F^{(1)} = C' \circ E^{(r')} \circ \ldots \circ E^{(1)} \circ C$$

for some affine bijections $C, C' : \mathbb{F}_2^N \to \mathbb{F}_2^N$. We also require that the quadratic maps arise from affine-quadratic self-equivalences as in the implicit white-box proposal.

The procedure consists of two steps:

1. Eliminating quadratic self-equivalence maps by using the affine-encoded addition inversion technique based on interpolation (Subsection 5.4).

2. Finding symbolically affine self-equivalences of bijective modular addition layers which resolve the intermediate linear maps into the Speck linear layer.

8.1 Eliminating quadratic self-equivalences

By combining all affine maps with quadratic maps, we obtain a sequence of alternating bijective modular addition S and quadratic maps $Q^{(i)}$ (note that here Q's are different than those from the previous section, as they include an affine map from the previous round):

$$S \circ Q^{(r)} \circ S \circ \ldots \circ S \circ Q^{(1)}.$$

The quadratic maps have structure $Q^{(i)} = A^{(i+1)} \circ L \circ B^{(i)}$, where $A^{(i+1)}$ is the affine part of self-equivalence from the next modular addition, $B^{(i)}$ is the quadratic part of self-equivalence from the previous modular addition, L is the cipher's linear layer[2] (see Section 3 and Figure 1). We have

$$Q^{(i)} \circ S = A^{(i+1)} \circ L \circ B^{(i)} \circ S = A^{(i+1)} \circ L \circ S \circ A^{(i)}.$$

In other words, the function $Q^{(i)} \circ S$ is affine-equivalent to S. Note that we have two-sided affine equivalence because $A^{(i+1)}$ is unknown, as we recover these maps up to a composition with canceling affine encodings. Yet, we can run the affine-equivalence attack from Section 6 on $Q^{(i)} \circ S$ to recover $A^{(i+1)} \circ L$ and $A^{(i)}$, up to affine self-equivalences of S. Effectively, this approach allows to propagate the quadratic parts of affine-quadratic self-equivalences through the addition into affine maps.

Remark 6. This step can be performed already after detaching the quadratic encoding by the attack from Section 7, and attaching it to the previous round with its quadratic encoding also detached. This means that there is no need to perform the affine-encoding decomposition of the remaining part of the current round separately, but instead it should be done when composed together with the next round's input quadratic encoding. Thus, we only need one call to the attack from Section 6 per round.

[2]Note that here we define self-equivalences with respect to the bijective modular addition, and not to to the full round function.

The complexity of this step is dominated by the decomposition of affine-encoded bijective modular addition (Section 6), which is $\mathcal{O}(n^4)$ time and $\mathcal{O}(n^3)$ queries. Note that the bilinear relations are already obtained during the process of quadratic encoding recovery (Subsection 7.3), therefore, the precomputation step is not needed and we can do optimized queries "offline" (without calling the actual white-box oracle anymore), so that the optimal effective time complexity of $\mathcal{O}(n^6)$ is reached.

8.2 Aligning affine-self equivalences to match the cipher's linear layer

After eliminating quadratic maps, we are left with alternating bijective modular additions and affine maps. This problem was already solved in [VRP22] using Gröbner basis. Here we propose an alternative technique based solely on linear algebra.

We know that the affine maps should be equal to the cipher's linear layer and key/constant additions. However, the affine maps are recovered up to affine self-equivalences of the surrounding additions. Thus, we need to solve the following equation (considering only the linear part):

$$L = U \circ C \circ V, \tag{23}$$

where U is the unknown input linear part of an affine self-equivalence of S, V is the unknown output linear part of (another) affine self-equivalence of S, C is a known linear map, L is the (known) cipher's linear map. By rewriting it as

$$L \circ V^{-1} - U \circ C = 0 \tag{24}$$

we get a linear equation on entries of the matrices of V^{-1} and U. However, the solution is not unique: from (23) it is clear that for every bijective V there exists a satisfying matrix U. Therefore, we need to add constraints of U, V being linear parts of affine self-equivalences of bijective modular addition. This can be done by enforcing the shape of the matrix of the linear part of an affine self-equivalence, established in [RVP22], which we parameterize in more details, based on experiments using our decomposition attack:

$$U = \left(\begin{array}{ccccc|ccccc} 1 & & & & & & & & & \\ t_1 & 1 & & & & & & & & \\ & & \ddots & & & & & & & \\ & & & 1 & & & & & & \\ t_2 & c & \cdots & c & t_3 & t_4 & c & \cdots & c & t_5 \\ \hline a_1 & & & & & b_2 & & & & \\ a_2 & & & & & b_3 & 1 & & & \\ \vdots & & & & & \vdots & & \ddots & & \\ a_{n-1} & & & & & b_{n-1} & & & 1 & \\ a_n & d & \cdots & d & t_6 & b_n & d & \cdots & d & t_7 \end{array}\right),$$

where the rulers divide the $N \times N$ matrix into four $n \times n$ matrices. In addition, V and V^{-1} have the same shape.

Putting these parameterized matrices into (24), we get a linear system on the undetermined entries, which experimentally always has a unique solution.

After the linear layer between the two additions is modified using recovered U, V maps into the cipher's linear layer, the linear parts of self-equivalences are fixed, and we are left with the constant parts. First, we recover all possible input/output constant additions that make the current oracles into precise bijective modular addition. This can be done efficiently in a bit-by-bit manner. Note that the round key in Speck is XORed right after the addition, so we extract the subkey as the output constant of the addition.

The recovered constants are defined up to additions in the most significant bits. Furthermore, a specific of our decomposition technique (ignored possible XORs from bit $n-2$ to $n-1$ in the first step) makes the $(n-2)$-nd bit of the addition undetermined. As a result, we recover the subkey excluding 2 most significant bits, which are negligible to search even for 4 rounds at once. Four consecutive round keys allow to easily undo the key schedule and recover the master key.

9 Experimental Evaluation

We implemented the full key recovery attack on an implicit white-box implementation of Speck32/64 and Speck64/128, generated using strongest parameters we could compile (affine-quadratic self-equivalences, degree-4 graph automorphisms, see Appendix A). All experiments were done on an Intel(R) Core(TM) i7-1185G7 3.00GHz CPU on a laptop (single core or parallel round decompositions distributed across cores). We remark that the implementation is a proof-of-concept and many optimizations are possible. The implementation is available at

https://github.com/cryptolu/implicit_ARX_whitebox_cryptanalysis

For Speck32/64 ($n = 16$), one white-box round call takes about 6 milliseconds. For comparison, an affine-encoded optimized oracle takes 15-20 microseconds (available after removing the quadratic encoding and interpolation). In about 30 minutes, we successfully decomposed all 20 rounds of a white-box instance (which we did to ensure that all rounds are susceptible to our methods). We skipped the first round as it (in principle) may have full MQ-like quadratic external encoding, it is unnecessary for key recovery. On average, each round decomposition takes 1.5 minutes. The unique master key candidate matches all the intermediate subkey candidates (taking into account the Speck's key schedule).

For Speck64/128 ($n = 32$), one white-box round call takes about 320 milliseconds. For comparison, an affine-encoded optimized oracle requires 45-50 microseconds. One decomposition attempt takes about 40-60 minutes (repeated attempts are needed when certain matrices occurring in the process are non-invertible, although only a part of the process is repeated). We decomposed the first 10 rounds (skipping the very first one with a possible external encoding), and recovered the unique master key candidate matching all the subkeys.

10 Conclusions

We would like to draw several conclusions from our work.

Algebraic attacks without one external encoding As current designs in the implicit function framework do not offer new kinds of encodings, they do not offer new protections against gray-box-like attacks in the pure white-box setting (without external encodings). In particular, quadratic encodings are defeated by a quadratic algebraic attack.

Implicit function framework Our generic interpolation-based optimization attack minimizes the degrees of the implicit functions describing a round function. Although the cost may be high (e.g., $\mathcal{O}(n^9)$ in the quadratic-affine encodings case), it is polynomial in the size of descriptions of the function (unless compression methods would be discovered in the future). This attack questions/limits the utility of obfuscation techniques for implicit functions (such as graph automorphisms). We remark that the implicit function framework itself remains a very interesting tool for implementing round functions and embedding encodings at a small cost, yet it currently lacks fitting primitives. We hope for more future work in this direction.

Implicit bilinear inversion Our inversion attack shows that round functions with bilinear implicit functions can not be used with affine encodings, and even quadratic-affine encodings require special care. Of course, the attack only targets the non-invertibility security goal (not claimed by [RVP22] but worth studying in general).

ARX-based round function Our main decomposition attacks show that an ARX round with single modular addition has many weaknesses. As mentioned above, it has a bilinear implicit function. Its quadratic self-equivalences (at least, the ones discovered by [RVP22]) are extremely sparse (only a few distinct monomials, up to affine equivalence).

Possible countermeasures As our attacks rely on the isolation and black-box access to each round (as is common in white-box attacks in the presence of external encodings). Code obfuscation can potentially prevent the attack by deterring a human reverse-engineer. An alternative direction is searching for cubic or denser quadratic self-equivalences of the bijective modular addition.

Acknowledgements

The work was supported by the Luxembourg National Research Fund's (FNR) and the German Research Foundation's (DFG) joint project APLICA (C19/IS/13641232).

A Generating Instances of White-box Speck

The authors of [RVP22] provided a Python/Sagemath [Sag22] implementation to generate instances of the Speck block cipher with their white-box construction, more specifically they provide direct support for Speck32/64, Speck64/128 and Speck128/256. Their implementation allows one to generate white-box instances of the Speck block cipher with different parameters, and thus we will quickly describe the most relevant ones as well as which ones we used, keeping in mind that our attacks are targeted to the most generic cases in their framework.

The first set of parameters allows to set several parts of the encodings as trivial, namely the affine part of the encodings, the quadratic part, the affine-quadratic equivalences used and the external encodings. As our goal is to provide generic attacks on their framework, the choice here is rather simple. The affine part and quadratic part of the encodings, as well as the affine-quadratic self equivalence relations, are all chosen as non-trivial. External encodings will be disabled (i.e. set to trivial) only in Section 4, and enabled (i.e. non-trivial) in the rest of the paper.

The second set of parameters influences the concrete white-box implementation, more specifically how the implicit functions are generated. At some point in the generation, graph automorphisms are needed (we refer the reader to the original paper for their role), which can be set to trivial with one parameter. It is a bit unclear how this influence the resulting instance, but to provide the most generic instances, we kept these as non-trivial. Note that it does seem to severely slow down the time needed to generate the white-box instance. A second parameter allows one to use so-called "redundant perturbations" in the instance based on the work of [BCD06]. The authors do not provide details about how this functionality works, however from our understanding, they only affect the actual implementation (i.e. resulting code) and not the resulting encoded round function. This means that assuming every other parameters are the same (i.e. same key, same encodings, same affine-quadratic equivalences etc.) for one instance $\overline{E}$ without these perturbations, and one other instance $\overline{E}'$ with these perturbations, we should always have $\overline{E}(x) = \overline{E}'(x)$,

as the perturbations seems to only influence *how* the encoded round function are evaluated on a given input, not the actual result[3]. As such, we chose to disable these perturbations.

Finally, as stated in the previous section, encoded round function are implemented as *implicit functions*, which, in the author's implementation, can be of degree 2, 3 or 4 depending on some parameters. From the authors (Section 6, bottom of page 24 in [RVP22]), an implicit function of degree 2 implies that no quadratic encodings are used, only affine encodings and affine self-equivalences. As we aim at attacking the generic case where the encodings are quadratic, this is not something we want so we do not allow degree 2 implicit functions. When the degree of implicit functions is set to either 3 or 4, quadratic encodings and affine-quadratic self-equivalences are used, however as stated by the authors, degree 3 implicit functions implies to choose carefully which quadratic functions are used, and as such might not be as generic as we would like. An additional parameters allows to enforce that *every* implicit function (i.e. encoded round function) is of the chosen degree. However, when we tried to enforce *every* implicit function to be of degree 4 (as it should be the most generic case), the generation of white-box instances failed and we were not able to fix their code to make it work. As such, for every instance we generated, we enforced the degree to be at least 3 and at most 4, which results in some encoded rounds to be of degree 3 and others to be of degree 4. As we will see in later sections, we were able to recover the round key of every round (both degree 3 and 4) without having to consider this degree, so we do not expect this to have any influence on our attacks.

Finally, a small implementation detail for our attacks. The authors' tool allows to generate instances and either use them directly in Python/Sagemath, or export them to some C code. While all our attacks are implemented in Python/Sagemath, using the direct Sagemath evaluation of encoded rounds turned out to be extremely slow. Thus our solution was to export the generated instance into C code, and then compile this C code into a shared library, which we can use from Python/Sagemath using the `ctypes` Python library. This proved to be a huge improvement for running our attacks, as a very large part of the time complexity (in practice) is due to the rather slow calls to the oracle (even with this shared library).

B Optimized Implementation of Implicit Bilinear Functions

Let a function $F(X) = Z$ be given implicitly by

$$P : \mathbb{F}_2^N \times \mathbb{F}_2^N \to \mathbb{F}_2^m : (X, Z) \mapsto \sum_{i,j} \lambda_{i,j}^{XZ} X_i Z_j \oplus \sum_i \lambda_i^X X_i \oplus \sum_i \lambda_i^Z Z_i \oplus \lambda,$$

where each λ-coefficient belongs to $\mathbb{F}_2^m$. It can be rewritten as

$$P(X,Z) = \sum_{i=0}^{N-1} X_i \cdot \Big(\sum_{j=0}^{N-1} \lambda_{i,j}^{XZ} Z_j \oplus \lambda_i^X\Big) \quad \oplus \quad \sum_i \lambda_i^Z Z_i \oplus \lambda \tag{25}$$

$$= \sum_{i=0}^{N-1} X_i \cdot A^{(i)} \times (Z_0, \ldots, Z_{N-1}, 1)^T \oplus B \times (Z_0, \ldots, Z_{N-1}, 1)^T, \tag{26}$$

$$= \left(\sum_{i=0}^{N-1} X_i \cdot A^{(i)} \oplus B\right) \times (Z_0, \ldots, Z_{N-1}, 1)^T. \tag{27}$$

where $A^{(i)}, B$ are $m \times (N+1)$-bit matrices. Therefore, the solution of P for Z can be computed as following:

[3] Checking the source code of their tool, it even seems that these perturbations can result in a failure to evaluate an encoded round function

1. Compute the matrix sum $C(X) = \sum_{i=0}^{N-1} X_i \cdot A^{(i)} \oplus B$.
2. Find the right kernel of C, which should consist of the only nonzero vector $(Z_0, \ldots, Z_{N-1}, 1)$ with $Z = F(X)$.

In practice, the matrix sum can be done faster by precomputing and storing w-block sums

$$T(X_{kw,\ldots,kw+w-1}) = \sum_{i=kw}^{kw+w-1} X_i \cdot A^{(i)},$$

reducing N matrix additions to N/w at the cost of storing $2^w N/w$ matrices. Rows of the matrices stored as CPU words, allowing efficient Gaussian elimination to recover the kernel, which is the dominant step of the procedure.

This method is equivalently efficient for computing the inverse of F using the same implicit function (with swapped roles of variables).

Implemented in C, this method executes 1 query for affine-encoded bijective modular addition with $n = 16$ in 20 microseconds on average on a laptop with an Intel(R) Core(TM) i7-1185G7 3.00GHz CPU.

References

[AABM20] Estuardo Alpirez Bock, Alessandro Amadori, Chris Brzuska, and Wil Michiels. On the security goals of white-box cryptography. *IACR TCHES*, 2020(2):327–357, 2020. https://tches.iacr.org/index.php/TCHES/article/view/8554. 97

[ABF+20] Estuardo Alpirez Bock, Chris Brzuska, Marc Fischlin, Christian Janson, and Wil Michiels. Security reductions for white-box key-storage in mobile payments. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 221–252. Springer, Heidelberg, December 2020. 97

[BCD06] Julien Bringer, Hervé Chabanne, and Emmanuelle Dottax. White box cryptography: Another attempt. *IACR Cryptol. ePrint Arch.*, page 468, 2006. 131

[BCH16] Chung Hun Baek, Jung Hee Cheon, and Hyunsook Hong. White-box aes implementation revisited. *Journal of Communications and Networks*, 18(3):273–287, 2016. 97

[BGEC04] Olivier Billet, Henri Gilbert, and Charaf Ech-Chatbi. Cryptanalysis of a white box AES implementation. In Helena Handschuh and Anwar Hasan, editors, *SAC 2004*, volume 3357 of *LNCS*, pages 227–240. Springer, Heidelberg, August 2004. 98, 99

[BHMT16] Joppe W. Bos, Charles Hubain, Wil Michiels, and Philippe Teuwen. Differential computation analysis: Hiding your white-box designs is not enough. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 215–236. Springer, Heidelberg, August 2016. 99, 102

[BU18] Alex Biryukov and Aleksei Udovenko. Attacks and countermeasures for white-box designs. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 373–402. Springer, Heidelberg, December 2018. 99, 102

[CD08] Nicolas Courtois and Blandine Debraize. Algebraic description and simultaneous linear approximations of addition in Snow 2.0. In Liqun Chen, Mark Dermot Ryan, and Guilin Wang, editors, *ICICS 08*, volume 5308 of *LNCS*, pages 328–344. Springer, Heidelberg, October 2008. 99, 108, 109

[CEJv03] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. White-box cryptography and an AES implementation. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002*, volume 2595 of *LNCS*, pages 250–270. Springer, Heidelberg, August 2003. 97, 99

[CEJvO03] Stanley Chow, Phil Eisen, Harold Johnson, and Paul C. van Oorschot. A white-box des implementation for drm applications. In Joan Feigenbaum, editor, *Digital Rights Management*, pages 1–15, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. 97

[CLO10] David A. Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer Publishing Company, Incorporated, 3rd edition, 2010. 127

[DFLM18] Patrick Derbez, Pierre-Alain Fouque, Baptiste Lambin, and Brice Minaud. On recovering affine encodings in white-box implementations. *IACR TCHES*, 2018(3):121–149, 2018. https://tches.iacr.org/index.php/TCHES/article/view/7271. 112, 113

[GPRW20] Louis Goubin, Pascal Paillier, Matthieu Rivain, and Junwei Wang. How to reveal the secrets of an obscure white-box implementation. *Journal of Cryptographic Engineering*, 10(1):49–66, April 2020. 99, 102

[KPG99] Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced Oil and Vinegar signature schemes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 206–222. Springer, Heidelberg, May 1999. 111, 126

[LM01] Helger Lipmaa and Shiho Moriai. Efficient algorithms for computing differential properties of addition. In *FSE*, volume 2355 of *LNCS*, pages 336–350. Springer, 2001. 107, 108

[LN05] H.E. Link and W.D. Neumann. Clarifying obfuscation: improving the security of white-box des. In *International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, volume 1, pages 679–684 Vol. 1, 2005. 97

[MGH09] Wil Michiels, Paul Gorissen, and Henk D. L. Hollmann. Cryptanalysis of a generic class of white-box implementations. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *SAC 2008*, volume 5381 of *LNCS*, pages 414–428. Springer, Heidelberg, August 2009. 98

[RP20] Adrián Ranea and Bart Preneel. On self-equivalence encodings in white-box implementations. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 639–669. Springer, Heidelberg, October 2020. 97, 99

[RVP22] Adrián Ranea, Joachim Vandersmissen, and Bart Preneel. Implicit white-box implementations: White-boxing ARX ciphers. In *CRYPTO (1)*, volume 13507 of *Lecture Notes in Computer Science*, pages 33–63. Springer, 2022. 98, 99, 100, 101, 102, 105, 110, 111, 121, 122, 124, 125, 126, 127, 129, 131, 132

[Sag22] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.7, Release Date: 2022-09-19)*, 2022. `https://www.sagemath.org`. 131

[VRP22] Joachim Vandersmissen, Adrián Ranea, and Bart Preneel. A white-box speck implementation using self-equivalence encodings. In *ACNS*, volume 13269 of *Lecture Notes in Computer Science*, pages 771–791. Springer, 2022. 98, 128, 129

[XL09] Yaying Xiao and Xuejia Lai. A secure implementation of white-box aes. In *2009 2nd International Conference on Computer Science and its Applications*, pages 1–6, 2009. 97