Implicit White-Box Implementations: White-Boxing ARX Ciphers

Adrián Ranea[1][0000−0002−8697−7423], Joachim Vandersmissen[2], and Bart Preneel[1][0000−0003−2005−9651]

[1] imec-COSIC, KU Leuven, Belgium
`firstname.lastname@esat.kuleuven.be`
[2] atsec information security
`joachim@atsec.com`

Abstract. Since the first white-box implementation of AES published twenty years ago, no significant progress has been made in the design of secure implementations against an attacker with full control of the device. Designing white-box implementations of existing block ciphers is a challenging problem, as all proposals have been broken. Only two white-box design strategies have been published this far: the CEJO framework, which can only be applied to ciphers with small S-boxes, and self-equivalence encodings, which were only applied to AES.

In this work we propose implicit implementations, a new design of white-box implementations based on implicit functions, and we show that current generic attacks that break CEJO or self-equivalence implementations are not successful against implicit implementations. The generation and the security of implicit implementations are related to the self-equivalences of the non-linear layer of the cipher, and we propose a new method to obtain self-equivalences based on the CCZ-equivalence. We implemented this method and many other functionalities in a new open-source tool `BoolCrypt`, which we used to obtain for the first time affine, linear, and even quadratic self-equivalences of the permuted modular addition. Using the implicit framework and these self-equivalences, we describe for the first time a practical white-box implementation of a generic Addition-Rotation-XOR (ARX) cipher, and we provide an open-source tool to easily generate implicit implementations of ARX ciphers.

Keywords: White-box cryptography · Self-equivalence · Implicit implementation · ARX

1 Introduction

In some settings, such as digital rights management (DRM) or mobile banking, an attacker might get full control of the device performing the sensitive cryptographic computations. This extreme case cannot be captured by traditional black-box cryptography, which considers an attacker with only access to the input and output behaviour of the cryptographic algorithm, or even grey-box cryptography, that assumes some leakage from the device can be extracted from an attacker with partial access.

This setting was captured into the so-called white-box model by Chow et al. [13]. In this cryptographic model, it is assumed that the attacker has full control on the device running the cryptographic algorithm. In other words, the attacker can observe and even modify all the intermediate values of the cryptographic computation.

Chow et al. also proposed in this seminal work a design of an implementation of AES with the ambitious goal of preventing key-extraction attacks without relying on secure hardware or on the secrecy of the design. The main idea of this design, later called the CEJO framework, is to represent the cipher as a network of encoded look-up tables, where each table is composed with random perturbations or encodings in such a way that the output encoding of one step cancels the next input encoding. The encodings are small functions such as 4-bit or 8-bit permutations to avoid huge look-up tables and implementations with impractical size. To prevent a trivial attack, the first and last encodings are not cancelled, and due to these external encodings the resulting white-box implementation is not functionally equivalent to the original cipher.

To avoid external encodings and to keep the input-output behaviour of the cipher, some industrial white-box implementations rely on the secrecy of their designs. However, the WhibOx Contest [40] has shown that these white-box implementations can be broken by automated white-box attacks based on side-channel analysis and fault attacks [4,10,35]. This work focuses on the white-box setting proposed by Chow et al. and white-box implementations relying on secret designs [9] are outside the scope of this paper.

Several CEJO implementations [3, 14, 23, 26, 27, 37, 43] have been proposed, but all of them have been broken. These attacks exploit some properties of the underlying cipher (AES in most cases), but efficient generic attacks have also been proposed to the CEJO framework [3, 17, 29], which can break CEJO implementations of a wide class of ciphers.

Apart from CEJO implementations only one other type of white-box implementations, of existing block ciphers and without relying on secret designs, has been published: self-equivalence implementations [28, 34]. In the latter implementations, the round functions are protected by affine self-equivalences of the non-linear layer S, that is, affine permutations (A, B) such that $S = B \circ S \circ A$ [15]. In other words, each round in a self-equivalence implementation is composed of the non-linear layer, which is left unprotected, and the encoded affine layer, which encodes the affine layer containing the round key material with self-equivalences of the non-linear layer.

Whereas CEJO implementations can only use small encodings, self-equivalence implementations can use large encodings and avoid many of the attacks to the CEJO framework. Ranea and Preneel showed in [34] that the affine self-equivalence group of the non-linear layer plays a major role in the security of self-equivalence and CEJO implementations, and they suggest securing a cipher with a non-linear layer that is composed of large S-boxes and that has large affine self-equivalence group, instead of securing AES which was the focus of most earlier work.

Unfortunately and apart from the insecure example for AES [28], no other self-equivalence implementation has been proposed this far. The main difficulty is finding a suitable non-linear layer. Among the functions used in non-linear layers, the power function $x \mapsto x^d$ is one of the few for which a large affine self-equivalence subgroup[3] is known for large bitsizes. However, power functions are used in block ciphers mostly in small S-boxes, with the exception of some algebraic ciphers [1].

One of the most important large non-linear permutations used in block ciphers is the permuted modular addition, $(x, y) \mapsto (x \boxplus y, y)$, but no previous work has studied its affine self-equivalences. However, quadratic functions tend to have many affine self-equivalences, and the modular addition is CCZ-equivalent to a quadratic function [36]. CCZ-equivalence [12] is an equivalence relation between functions based on their graphs that preserves many properties such as differential or linear properties. Thus, the quadratic CCZ-equivalence of the modular addition suggests its affine self-equivalence group can be very large.

1.1 Contributions

In this work we first address the problem of finding the affine self-equivalence group of the permuted modular addition. Schulte-Geers obtained in [36] the differential and linear properties of the modular addition from the differential and linear properties of its simpler CCZ-equivalent quadratic function. Inspired by this approach and the relation between the self-equivalences of two CCZ-equivalent functions, we propose a method to obtain the self-equivalences of a non-linear function from a low-degree CCZ-equivalent function. Our method can find affine but also other types of self-equivalences, such as self-equivalences (A, B) with A being affine and B being quadratic or affine permutations that map the graph of a function to itself.

We provide a new open-source library `BoolCrypt`[4] that implements the previous method and many functionalities related to vectorial Boolean functions, self-equivalences, and functional equations. We dedicated a significant engineering effort to equip `BoolCrypt` with plenty of functionalities, a modular structure and an extensive documentation, so that it can be useful and practical for the community.

After running `BoolCrypt` on the permuted modular addition, we obtained subsets of $3 \times 2^{2n+2}$ linear, $3 \times 2^{2n+8}$ affine, and $3^2 \times 2^{3n+14} - 3 \times 2^{2n+8}$ affine-quadratic self-equivalences for wordsize $n \in \{4, 5 \ldots, 64\}$. It is worth mentioning that the self-equivalence results, the tool `BoolCrypt` and the generic method to

[3] The power function $F(x) = x^d \in \mathbb{F}_{2^n}$ has at least $n(2^n - 1)$ linear self-equivalences of the form $(A(x), B(x)) = (ax^{2^i},\ a^{-d2^{n-i}} x^{2^{n-i}})$, where $a \neq 0$ and $i = 0, 1, \ldots, n-1$. The whole linear self-equivalence group has only been found for some exponents [41].

[4] https://github.com/ranea/BoolCrypt

compute self-equivalences can be of independent interest for other areas apart from white-box cryptography.

Unfortunately, the self-equivalences that we found are too structured and not suitable for self-equivalence implementations. Despite the exponential number of these linear, affine and affine-quadratic self-equivalences, they have a common sparse shape and low-entropy constant vectors. As a result, the round keys are not securely hidden in the constant vectors of the encoded affine layers.

Thus, we propose *implicit implementations*, a new design of white-box implementations that combines large self-equivalences with large affine encodings to prevent attacks exploiting the structure of the self-equivalences. Implicit implementations represent each round function by a low-degree *implicit* function and apply the large encodings on the implicit round functions.

Whereas CEJO implementations can only apply small encodings and self-equivalence implementations can only use self-equivalence encodings, the main advantage of implicit implementations is that they can efficiently encode the low-degree implicit round functions with large affine permutations and even large non-linear self-equivalences.

We analyse the security of implicit implementations against key-extraction attacks and show that implicit implementations that use non-linear encodings or where the non-linear layer of the cipher is not composed of small S-boxes prevent all known generic key-extraction attacks. We also propose a new generic attack for the implicit framework that provides insightful requirements for implicit implementations with affine encodings.

Using the implicit framework and the self-equivalences of the permuted modular addition, we describe an implicit implementation of a generic cipher with the permuted modular addition as the non-linear layer, which captures the well-known family of Addition-Rotation-XOR (ARX) ciphers. We analyse the security of these implicit implementations and provide an open-source tool[5] to generate implicit implementations of ARX ciphers in an automated way.

Designing white-box implementations of existing ciphers is currently the most challenging problem in white-box cryptography. Since the first white-box implementation of AES published twenty years ago [13], no major progress has been made in the design of white-box implementations; only CEJO implementations with small encodings and a toy self-equivalence implementation of AES were proposed thus far. In this work we address this challenging problem by proposing the new design framework of implicit implementations, which not only prevent all known generic key-extraction attacks but also can be applied to ARX ciphers for the first time.

Outline. In Sect. 2 the notation and the preliminaries are introduced. Implicit implementations are presented in Sect. 3, and their security is analysed in Sect. 4. In Sect. 5 we propose a method to obtain self-equivalences based on a low-degree CCZ-equivalent function, and we show the results of this method applied to the permuted modular addition. We then describe an implicit implementation of a

[5] https://github.com/ranea/whiteboxarx

generic ARX cipher in Sect. 6, together with its security analysis and the tool to generate implicit implementations of ARX ciphers. Section 7 presents the conclusions and future work.

2 Preliminaries

Let $\mathbb{F}_2^n$ be the vector space with n-bit values. We denote the addition in $\mathbb{F}_2^n$ by $\oplus$, and the function representing the addition by a constant k by $\oplus_k : x \mapsto x \oplus k$. The identity function in $\mathbb{F}_2^n$ is denoted by Id_n. Functions from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ will be called (n, m)-bit functions or just n-bit functions if $m = n$. Given two functions F and G, their composition is denoted by $F \circ G$ and their concatenation by $(F, G)(x, y) = (F(x), G(y))$.

The degree of an (n, m)-bit function F refers in this paper to the algebraic degree of F, that is, the maximum polynomial degree of the m multivariate polynomials uniquely representing the coordinate Boolean functions of F. A function F is affine if its algebraic degree is 1, and it is linear if, in addition, $F(0) = 0$. A non-linear function is a function with algebraic degree greater than or equal to 2. In particular, functions with algebraic degree 2, 3, and 4 are called quadratic, cubic and quartic functions respectively.

In this paper, functions are denoted by capital letters (e.g., F, G), elements of $\mathbb{F}_2^n$ by lowercase letters (e.g., x, y) and sets of functions by calligraphic letters (e.g., $\mathcal{F}, \mathcal{G}$).

2.1 Implicit Functions, Self-Equivalences and Graph Automorphisms

In this section we will introduce the three main ingredients of our new design of white-box implementations: implicit functions, self-equivalences, and graph automorphisms.

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

$$P(x, y) = 0 \iff y = F(x)\,.$$

Moreover, the n-bit variable vectors x and y will be called the *input* and *output variable vectors*, respectively.

For example, $P(x, y) = y \oplus F(x)$ is an implicit function for any F. An implicit function can also be defined as a function that implicitly defines the graph of a function. Let $\Gamma_F = \{(x, F(x)) : x \in \mathbb{F}_2^n\}$ be the graph of an n-bit function F. Then, P is an implicit function of F if and only if $\Gamma_F = \{(x, y) : P(x, y) = 0\}$. The next lemma describes how the composition of functions translates to their implicit functions and their graphs.

Lemma 1. *Let P be a $(2n, m)$-bit implicit function of an n-bit function F, (A, B) be a pair of n-bit permutations, U be a $2n$-bit permutation and V be an m-bit permutation with $V(0) = 0$.*

(i) $P' = P \circ (A, B^{-1})$ *is an implicit function of* $F' = B \circ F \circ A$.
(ii) $V \circ P$ *is an implicit function of* F.
(iii) $U(\Gamma_F) = \{U(x, F(x)) : x \in \mathbb{F}_2^n\}$ *is implicitly defined by* $P \circ U^{-1}$*, that is,*

$$U(\Gamma_F) = \{(x', y') : (P \circ U^{-1})(x', y') = 0\}\,.$$

Proof. The first two statements follows from the definition of P,

$$\big(V \circ P \circ (A, B^{-1})\big)(x, y) = 0 \iff P(A(x), B^{-1}(y)) = V^{-1}(0) = 0$$
$$\iff B^{-1}(y) = F(A(x)) \iff y = B(F(A(x)))\,,$$

and similarly for the last statement,

$$U(\Gamma_F) = \{U(x, y) : P(x, y) = 0\} = \left\{ (x', y') : \begin{array}{c} (x', y') = U(x, y) \\ P(x, y) = 0 \end{array} \right\}. \quad \square$$

In other words, applying a permutation A to the input of F and a permutation B to the output of F corresponds to applying (A, B^{-1}) to the input of an implicit function of F. Moreover, applying any permutation V with the fixed point 0 to an implicit function of F leads to another implicit function of F. The third statement of Lemma 1 shows that applying a permutation U to the points of the graph of F leads to the set of points defined by an implicit function of F composed with the inverse of U.

A *self-equivalence* of a function F is a pair of permutations (A, B) such that $F = B \circ F \circ A$. If (A, B) is a self-equivalence of F, we say that A (resp. B) is a right (resp. left) self-equivalence of F. Moreover, if A and B are affine (resp. linear), we say that (A, B) is an *affine (resp. linear) self-equivalence*. In this work we also consider non-linear self-equivalences; if A is affine and B is quadratic we say that (A, B) is an *affine-quadratic self-equivalence*.

The set of self-equivalences forms a group with respect to the composition, and the subsets of affine and linear self-equivalences are subgroups respectively [19, 25]. Moreover, given a function F and two permutations C and D, the self-equivalence groups of F and $D \circ F \circ C$ are conjugates, that is,

$$\begin{aligned} &\big(A, B\big) \text{ is a self-equivalence of } F \iff \\ &\qquad \big(C^{-1} \circ A \circ C, D \circ B \circ D^{-1}\big) \text{ is a self-equivalence of } D \circ F \circ C\,. \end{aligned}$$

A *graph automorphism*[6] of F is a permutation U such that $\Gamma_F = U(\Gamma_F)$. It is easy to show that the set of graph automorphisms is a group with respect to the composition, and its restriction to affine permutations is also a subgroup. The next lemma shows the relation among implicit functions, self-equivalences and graph automorphisms.

Lemma 2. *Let* P *be an implicit function of* F*. If* (U, V) *is a self-equivalence of* P *with* $V(0) = 0$*, then* U *is a graph automorphism of* F*. Moreover,* (A, B) *is a self-equivalence of* F *if and only if* $U(x, y) = (A(x), B^{-1}(y))$ *is a graph automorphism of* F*.*

[6] Self-equivalences are sometimes called automorphisms in the literature, but in this paper we only use the term automorphism to refer to a graph automorphism.

Proof. From Lemma 1, we have that $P' = V \circ P$ is an implicit function of F. Moreover,

$$0 = (P' \circ U)(x, y) \iff 0 = V(0) = V(P(U(x, y)) = P(x, y) \iff y = F(x)\,.$$

Thus, $P' \circ U$ is an implicit function of F and from Lemma 1, $U^{-1}(\mathit{\Gamma}_F) = \mathit{\Gamma}_F$. Since the set of graph automorphisms is a group, U is also a graph automorphism of F.

To prove the last statement, let A, B and U be three permutations and let P be an implicit function of F. For the forward direction, note that if (A, B) is a self-equivalence of F, then

$$(P \circ (A, B^{-1}))(x, y) = 0 \iff y = B(F(A(x))) = F(x)\,.$$

In other words, $P \circ (A, B^{-1})$ is another implicit function of F. Reasoning as in the previous case, $U = (A, B^{-1})$ is a graph automorphism of F. For the other direction, if $U = (A, B^{-1})$ is a graph automorphism of F, then $P \circ U^{-1}$ and $P \circ U$ are implicit functions of F (Lemma 1), which proves

$$y = F(x) \iff (P \circ (A, B^{-1}))(x, y) = 0 \iff y = B(F(A(x)))\,. \quad \square$$

In other words, the self-equivalences of a function F and some right self-equivalences of their implicit functions can be embedded in the group of graph automorphisms of F. In general, the group of graph automorphisms of F contains more elements, and some of them do not correspond to self-equivalences of F or self-equivalences of an implicit function of F. The next lemma shows that a function can have multiple implicit functions; its proof is straightforward from Lemmas 1 and 2.

Lemma 3. *Let P be an implicit function of F. If V is a permutation with $V(0) = 0$ and U is a graph automorphism of F, then $V \circ P \circ U$ is an implicit function of F.*

2.2 Encoded Implementations

Given an n-bit iterated block cipher, we denote the encryption function for a fixed key k by $E_k = E^{(n_r)}_{k^{(n_r)}} \circ E^{(n_r-1)}_{k^{(n_r-1)}} \circ \cdots \circ E^{(1)}_{k^{(1)}}$, where $E^{(i)}_{k^{(i)}}$ denotes the ith round function and $k^{(i)}$ denotes the ith round key. For ease of notation, we omit the round-key subscript of the round functions.

CEJO and self-equivalence implementations can be seen as encoded implementations, a class of white-box implementations based on the notion of encoding.

Definition 1 ([13]). *Let F be an (n, m)-bit function and let (I, O) be a pair of n-bit and m-bit permutations, respectively. The function $\overline{F} = O \circ F \circ I$ is called an encoded F with input and output encodings I and O, respectively.*

Definition 2 ([34]). *Let $E_k = E^{(n_r)} \circ E^{(n_r-1)} \circ \cdots \circ E^{(1)}$ be the encryption function of an iterated n-bit block cipher with fixed key k. An encoded (white-box) implementation of E_k is an encoded E_k composed of encoded round functions, that is,*

$$\overline{E_k} = \overline{E^{(n_r)}} \circ \cdots \circ \overline{E^{(1)}} = (O^{(n_r)} \circ E^{(n_r)} \circ I^{(n_r)}) \circ \cdots \circ (O^{(1)} \circ E^{(1)} \circ I^{(1)}) \,,$$

where the round encodings $(I^{(1)}, O^{(1)}), (I^{(2)}, O^{(2)}), \ldots, (I^{(n_r)}, O^{(n_r)})$ are n-bit permutation pairs s.t. $I^{(i+1)} = \left(O^{(i)}\right)^{-1}$ for $i = 1, 2, \ldots, n_r - 1$.

In other words, an encoded implementation is the composition of encoded round functions where the intermediate round encodings are cancelled out. Thus, $\overline{E_k}$ can also be written as $\overline{E_k} = O^{(n_r)} \circ E_k \circ I^{(1)}$, where the encodings $(I^{(1)}, O^{(n_r)})$ are also called the *external encodings*.

Definition 2 only considers round encodings satisfying the cancellation rule $I^{(i+1)} = \left(O^{(i)}\right)^{-1}$. However, in this paper we extend the definition of round encodings of an encoded implementation to any n-bit permutations pairs $(I^{(1)}, O^{(1)})$, $(I^{(2)}, O^{(2)}), \ldots, (I^{(n_r)}, O^{(n_r)})$ that satisfy the cancellation rule

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

for $i = 2, 3, \ldots, n_r - 2$. In Sect. 6 we will describe an example of round encodings satisfying the general cancellation rule given by Eqn. (1) but not the cancellation rule $I^{(i+1)} = \left(O^{(i)}\right)^{-1}$.

3 Implicit White-Box Implementations

In this section we present implicit implementations, a new class of white-box implementations of iterated block ciphers. The main idea of implicit implementations is to represent the high-degree round function of the cipher by a quasilinear implicit function of low degree.

Definition 3. *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.*

Informally, an implicit function is quasilinear if it is affine in the output variable vector. For any function F, a trivial quasilinear implicit function is $P(x, y) = y \oplus F(x)$. An example of a non-trivial quasilinear implicit function is the quadratic implicit function $P(x, y) = (x(xy+1), y(xy+1))$ of the finite field inversion over $\mathbb{F}_{2^n}$, i.e., $F(x) = x^{2^n-2}$.

The quasilinear property is necessary to make the implicit evaluation of F practical. Since a quasilinear implicit function becomes an affine function when the input variable vector x is fixed to a constant x_0, one can *implicitly* evaluate $F(x_0)$ by solving the affine system $P(x_0, y) = 0$ for y. Since the affine system $P(x_0, y) = 0$ has a unique y solution due to the definition of an implicit function, a $(2n, m)$-bit quasilinear implicit function of an n-bit function has at least n Boolean components ($m \geq n$).

With the notions of encoding, implicit and quasilinear function previously defined, we can now present implicit white-box implementations.

Definition 4. *Let $E_k = E^{(n_r)} \circ E^{(n_r-1)} \circ \cdots \circ E^{(1)}$ be the encryption function of an iterated n-bit block cipher with fixed key k, and let $\overline{E_k} = \overline{E^{(n_r)}} \circ \overline{E^{(n_r-1)}} \circ \cdots \circ \overline{E^{(1)}}$ be an encoded implementation of E_k. An implicit (white-box) implementation of $\overline{E_k}$ is a set of quasilinear implicit functions $\{P^{(1)}, P^{(2)}, \ldots, P^{(n_r)}\}$ where $P^{(i)}$ is an implicit function of $\overline{E^{(i)}}$.*

In other words, an implicit implementation is an alternative representation of an encoded implementation where the high-degree encoded round functions are given by low-degree implicit functions.

An implicit implementation is evaluated by implicitly evaluating the encoded round functions. Thus, given the output x_0 of the round $i-1$, the output y of the ith round is computed by finding the solution of the affine system $P^{(i)}(x_0, y) = 0$ for y. Note that an implicit implementation has the same input-output behaviour as its underlying encoded implementation.

The external encodings of an implicit implementation refer to the external encodings of its underlying encoded implementation. In this we work we focus on implicit implementations with non-trivial external[7] encodings.

The size of an implicit implementation mainly depends on the degree of the implicit encoded round functions $P^{(i)}$, which are implemented as binary multivariate polynomials. If the $(2n, m)$-bit functions $P^{(i)}$ have degree d, each one has at most

$$\sum_{i=0}^{d} \binom{n}{i} + n \sum_{i=0}^{d-1} \binom{n}{i}$$

coefficients, and thus the size of an implicit implementation is $\mathcal{O}(n^d)$. The running time of an implicit implementation is dominated by obtaining each affine system, by iterating over the monomials, in $\mathcal{O}(n^d)$ and by solving each affine system with Gaussian elimination in $\mathcal{O}(m^3)$.

3.1 Quasilinear Implicit Round Functions

While it is known how to obtain low-degree implicit functions of small S-boxes [21] and power mappings [7], obtaining a low-degree quasilinear implicit function of an arbitrary function (if it exists) is a hard problem. We will see how we can derive a low-degree quasilinear implicit function of the whole round function from a known low-degree quasilinear implicit function of the non-linear layer.

[7] It is worth to mention that all known white-box implementations of existing ciphers that do not rely on secret designs include external encodings in their designs. While external encodings impose severe limitations on the applicability of the white-box implementation, it is currently the only alternative to secret designs.

Assume the encoded ith round function is of the form

$$\overline{E^{(i)}} = O^{(i)} \circ (L \circ S \circ \oplus_{k^{(i)}}) \circ I^{(i)} \,,$$

where L is an affine permutation also called the linear layer, S is a non-linear permutation also called the non-linear layer, $k^{(i)}$ is the round key and $(I^{(i)}, O^{(i)})$ are the round encodings with $O^{(i)}$ being affine. Note that this approach to obtain a quasilinear implicit function of the ith round function applies to all rounds $i = 1, 2, \ldots, n_r$, and thus the affine restriction also applies to the external output encoding.

The output encoding $O^{(i)}$ needs to be affine for the quasilinear property, but there is no degree restriction on $I^{(i)}$. The degree of $I^{(i)}$ can be different from the degree of $O^{(i)}$ if the round encodings satisfy the cancellation rule given in Eqn. (1) rather than the one from Definition 2. For example, in Sect. 6 we describe an implicit implementation with quadratic input encodings and affine output encodings.

Given a known quasilinear implicit function T of S and by applying Lemma 1, we can obtain a quasilinear implicit function of $\overline{E^{(i)}}$ as

$$T \circ (\oplus_{k^{(i)}}, L^{-1}) \circ (I^{(i)}, (O^{(i)})^{-1}) \,.$$

Note that the left composition by affine functions and the right composition by diagonal functions $A(x, y) = (A_1(x), A_2(y)))$ with A_2 affine preserves the quasilinear property.

Moreover, we can replace the known implicit function of S with a different implicit function by applying Lemma 3. In particular, if V is a linear permutation and U is an affine graph automorphism of S that preserves the quasilinear property (i.e., $T \circ U$ is quasilinear), then $V \circ T \circ U$ is another quasilinear implicit function of S. However, U should not be a right self-equivalence of T to prevent U from being cancelled. That is, if U is a right self-equivalence of T, then $V \circ T \circ U = V' \circ T$ for some V' affine.

To conclude, given a quasilinear implicit function T of the non-linear layer S, we can build a quasilinear implicit function $P^{(i)}$ of the encoded round function $\overline{E^{(i)}} = O^{(i)} \circ (L \circ S \circ \oplus_{k^{(i)}}) \circ I^{(i)}$ by sampling a linear permutation $V^{(i)}$ and a quasilinear-preserving graph automorphism $U^{(i)}$ that is not a right self-equivalence of T, obtaining

$$P^{(i)} = V^{(i)} \circ T \circ U^{(i)} \circ (\oplus_{k^{(i)}}, L^{-1}) \circ (I^{(i)}, (O^{(i)})^{-1}) \,. \tag{2}$$

Figure 1 depicts a representation of $\overline{E^{(i)}}$ and $P^{(i)}$.

From Eqn. (2) it is easy to see that degree of $P^{(i)}$ only depends on the degree of T and the degree of the input encoding $I^{(i)}$, since the other functions are either linear or affine. The function T at least has degree 2 and n components if S is an n-bit non-linear function. Thus, we can consider affine or quadratic $I^{(i)}$ to build a practical implicit implementation.

Whereas CEJO or self-equivalence implementations impose severe restrictions on the encodings, an implicit implementation can use large affine or quad-

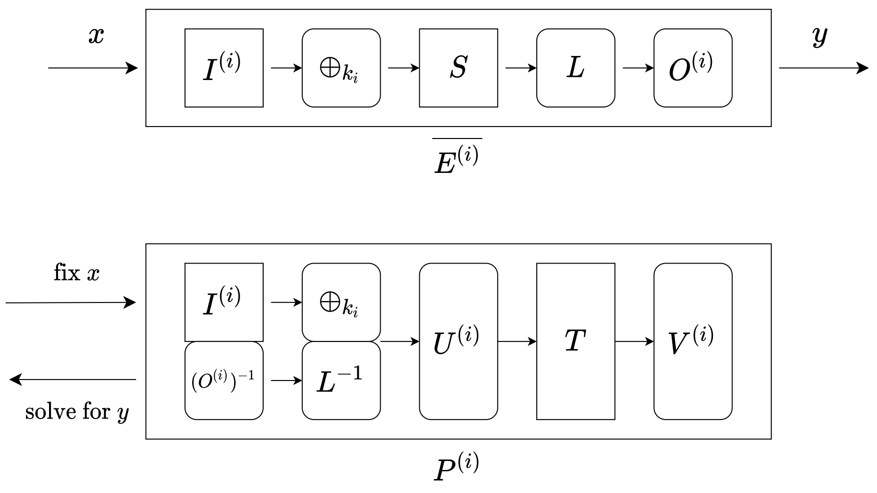

Fig. 1. An encoded round function $\overline{E^{(i)}}$ (top) given by the round function $(L \circ S \circ \oplus_{k^{(i)}})$ and the round encodings $(I^{(i)}, O^{(i)})$, and an implicit function $P^{(i)}$ (bottom) of $\overline{E^{(i)}}$, where T is an implicit function of the non-linear layer S, $U^{(i)}$ is a graph automorphism of S and $V^{(i)}$ is a linear permutation.

ratic encodings and have a practical[8] size and running time. As we will see in the next section, these encodings are the reason why implicit implementations prevent the white-box attacks that break CEJO or self-equivalence implementations.

4 Security Analysis

In this section, we analyse the security of implicit implementations against key-extraction attacks in the white-box model, where it is assumed that the specifications of the block cipher and the specifications of the implicit implementation are public, and the attacker is in possession of an implicit implementation but does not know the encodings or the key.

The security objective to resist key-extraction attacks in the white-box model was proposed by Chow et al. in their seminar work [13], and properly defined by Delerablée et al. as the unbreakability security notion [16]. For simplicity, in this work we will use the informal security objective of key-extraction resistance from [13], but we refer the reader to [16] for the complete definition of the unbreakability notion.

Key-extraction resistance or unbreakability is the minimum security goal a white-box implementation should aim for. In many applications, a white-box

[8] We will provide specific numbers for the size of an example of an implicit implementation in Sect. 6.

implementation of an encryption function should also prevent an attacker to obtain the corresponding decryption function. This security notion was formalized as strong white-box security in [6] and as one-wayness in [16]. However, one-wayness is much stronger than unbreakability, as building a white-box implementation of a block cipher satisfying the one-wayness notion involves turning the secret-key cipher into a public-key one. Other security notions for white-box implementations have also been proposed such as weak white-box security [6] or incompressibility [16], and traceability [16].

Due to the huge difficulty to design a white-box implementation of an existing cipher, CEJO and self-equivalence implementations [3,14,23,26–28,34,37,43] only aim at preventing key-extraction attacks and so do implicit implementations.

White-box implementations of existing block ciphers argue their key-extraction resistance by showing that all known key-extraction attacks cannot recover the key from the implementation faster than brute-force search, similar as how block ciphers argue their security in the black-box model.

4.1 Previous Generic Attacks

In this section we will show that all known generic key-extraction attacks, that is, key-extraction attacks to white-box implementations that do not exploit specific properties of the underlying ciphers, cannot recover the key from a generic implicit implementation if quadratic round input encodings are used or the non-linear layer is not composed of smaller S-boxes.

Note that showing that implicit implementations prevent all known generic key-extraction attacks is not sufficient to claim that a specific implicit implementation of a particular cipher is secure for two reasons. First, since the implicit framework is a new method, previous generic attacks were not designed to break implicit implementations. Thus, further research on new generic attacks targeting the implicit framework is needed. Second, for a specific implicit implementation of a particular cipher one still needs to check key-extraction attacks exploiting properties of the underlying cipher. Despite these two remarks, the fact that implicit implementations prevent all known generic key-extraction attacks is noteworthy as all CEJO and self-equivalence implementations have been broken even with generic attacks.

CEJO attacks. The first generic attack that we will consider is the reduction subroutine that transforms the encodings of a CEJO implementation to self-equivalence encodings. This subroutine was proposed in [34] and combines the three generic CEJO attacks proposed in [3, 17, 29]; showing that the reduction subroutine does not succeed implies that neither the three generic CEJO attacks do.

Since the reduction subroutine only requires black-box access to three consecutive encoded rounds, it can be applied to an implicit implementation. For an implicit implementation where the non-linear part of the round encodings is composed of m_{nl}-bit functions and the non-linear layer of the underlying cipher is composed of m_s-bit S-boxes, the complexity of the reduction subroutine is roughly $\mathcal{O}(2^{\max(3m_{nl}, m_s)})$ (Proposition 1 of [34]). For example, the reduction subroutine is efficient for an implicit implementation of AES ($m_s = 8$) with affine round encodings. However, the reduction subroutine is not efficient if the implicit implementation uses non-linear encodings or if the non-linear layer of the cipher is not composed of small S-boxes.

The CEJO framework cannot implement large non-linear encodings or such a non-linear layer due to its impractical size; a CEJO implementation requires roughly $\mathcal{O}(2^{\max(2m_{nl}, m_s)})$ bits of memory for each round [3]. On the other hand, the implicit framework can provide practical implementations under these constraints; the size of an implicit implementation is exponential in the degree of the implicit round functions, and not on the bitsize of the encodings or the non-linear layer. An example of a practical implicit implementation with large non-linear encodings and where the non-linear layer is composed of a single S-box will be described in Sect. 6.

Self-equivalence attacks. The generic attack to self-equivalence implementations proposed in [34] cannot be applied to implicit implementations since it requires access to encoded affine layers where the round encodings are self-equivalences of the non-linear layer. However, implicit implementations use random affine permutations for the round encodings, and the affine layer of the cipher is merged with the non-linear layer in the implicit round functions. Moreover, this attack also assumes the non-linear layer is composed of small S-boxes.

Self-equivalence implementations might also be vulnerable if the self-equivalences of the non-linear layer are very structured, such as having the same coefficient in some monomial terms for all self-equivalences. In this case, some coefficients in the encoded affine layers might directly leak coefficients from the round encodings or even bits from the round keys. Self-equivalence implementations of ciphers where the non-linear layer is the permuted modular addition exhibit this weakness, since the self-equivalences of the permuted modular addition have a common sparse shape and low-entropy constant vectors as we will see in Sect. 5.2. Implicit implementations do not exhibit this weakness since the round encodings use random and large affine permutations.

Automated white-box attacks. For white-box implementations not relying on external encodings but on secret designs, several automated white-box attacks have been proposed such as Differential Computational Analysis (DCA) or Differential Fault Analysis (DFA) [4,10,35]. These attacks easily break CEJO implementations in a fully automated way, but they assume the external encodings are the identity functions. The only automated attack successful against non-trivial external encodings is a DFA attack by Amadori et al. [2] to a CEJO implementation of AES where the external output encoding is the composition of 8-bit functions.

Since the implicit framework uses at least large affine permutations for the external encodings, current automated attacks cannot be applied to generic implicit implementations. On top of that, these attacks usually target an intermediate computation where the output is encoded with an small function and depends on a few key bits. However, in implicit implementations the only computations are evaluations of polynomials with large inputs and large encodings, and the outputs of these computations depend on the whole round keys. Finding new automated attacks to implicit implementations is an interesting challenge that we leave as future work.

4.2 Reducing Implicit Implementations to Self-Equivalence Implementations

We have shown that implicit implementations that use non-linear encodings or where the non-linear layer of the cipher is not composed of small S-boxes are secure against all known generic key-extraction attacks[9]. For the rest of this section, we will describe a new generic attack for implicit implementations.

The crucial step in most generic key-extraction attacks on encoded implementations is the reduction subroutine, that is, obtaining new encoded rounds with the same round key material but with simpler encodings [34]. Thus, we will not describe a complete key-extraction attack but a reduction subroutine that transforms an implicit implementation to a self-equivalence implementation, similar to the reduction subroutine from [34].

The reasons to consider a reduction to a self-equivalence implementation are three-fold. First, we will see that the self-equivalences of the non-linear layer play an important role in the security of implicit implementations, similar as in self-equivalence implementations. Second, if an implicit implementation is reduced to a self-equivalence one, then the generic attack to self-equivalence implementations [34] can be applied. Third, since a self-equivalence implementation is more efficient than an implicit one, an implicit implementation that can be reduced to a self-equivalence one does not provide any advantage.

Note that both CEJO and self-equivalence implementations can be transformed efficiently into an implicit one. First, any CEJO implementation can be transformed into a self-equivalence one using the techniques from [34]. Second, a self-equivalence implementation can be transformed into an implicit one simply by composing the encoded affine layers with the implicit function of the non-linear layer.

For our reduction subroutine, we will consider an intermediate round of an implicit implementation. Let $E = L \circ S \circ \oplus_k$ be an intermediate round function, $\overline{E} = O \circ E \circ I$ be an encoded function of E, and

$$P = U \circ T \circ V \circ (\oplus_k, L^{-1}) \circ (I, O^{-1})$$

be an implicit function of $\overline{E}$, as defined in Eqn. (2). For simplicity, we will drop the round index i. We will assume that the round key k, the encodings I and O,

[9] 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.

and the functions U and V are unknown to the adversary, but the linear layer L, the non-linear layer S, the implicit function T of S and the implicit function P are known.

The first step in most reduction subroutines is to solve a functional equation involving the round encodings. For an implicit implementation, the adversary can consider the two following functional equations:

$$P = Y \circ T \circ X \tag{3}$$

$$\overline{E} = Y \circ (L \circ S) \circ X\,. \tag{4}$$

In Eqn. (3), P and T are fully known, X is an unknown permutation with the same degree as I and Y is an unknown linear permutation. In Eqn. (4), $(L \circ S)$ is fully known, $\overline{E}$ can only be evaluated implicitly, X is an unknown permutation with the same degree as I and Y is an unknown affine permutation.

When the unknowns X and Y are affine permutations, this type of functional equations has received multiple names in the literature: the affine equivalence [8], the Isomorphism of Polynomials (IP) [31] or the affine-substitution-affine (ASA) [6, 30] problems. Most algorithms to solve affine equivalence problems exploit the particular structure of the central map. For example: (1) if T is a triangular function, the structural attack by [42] can solve Eqn. (3) in polynomial time; (2) for many quadratic T, Gröbner-based attacks have solved Eqn. (3) in polynomial time [19]; or (3) if S is composed of small m_s-bit S-boxes, the algorithm by Derbez et al. can solve Eqn. (4) in about $\mathcal{O}(2^{2m_s})$ [17].

When the unknown X is a non-linear permutation, this type of functional equations has been less studied in the literature, but some efficient algorithms have been proposed for particular quadratic central maps. Some examples are attacks on public-key schemes with 2 rounds [32], attacks on the ASASA structure [30] or attacks based on the decomposition of multivariate polynomials [20]. For the rest of the attack we will assume that I and X are affine permutations and we will leave the non-linear case for future work.

Obtaining any solution of Eqn. (4) allows the reduction from the implicit round function into an encoded affine layer. An arbitrary solution of Eqn. (4) for (X, Y) is of the form

$$(A \circ (\oplus_k \circ I), O \circ B)\ ,$$

where (A, B) is an affine self-equivalence of $L \circ S$. By combining a solution of that round and of the previous round, one can easily derive an encoded affine layer of that round. Repeating this process for all rounds leads to a complete reduction to a self-equivalence implementation.

On the other hand, not all the solutions of Eqn. (3) allow the reduction to an encoded affine layer. Every solution (X, Y) of Eqn. (3) is of the form

$$\left(A \circ (U \circ (\oplus_k, L^{-1}) \circ (I, O^{-1})), V \circ B\right)\ ,$$

where (A, B) is an affine self-equivalence of T. If (X, Y) is a solution with X diagonal (i.e., $X(x, y) = (X_1(x), X_2(y))$) and Y linear, then $A \circ U$ is diagonal and contains a self-equivalence of S by Lemma 2. Therefore, by combining two solutions of this form for that round and the previous round respectively, an adversary can easily build an encoded affine layer for that round.

As a result, an implicit implementation with affine encodings must ensure that no solution of Eqn. (4) can be obtained efficiently. Previously, we showed that an implicit implementation with affine encodings of a cipher with small S-boxes is vulnerable to the reduction subroutine of [34]. Similarly, such an implicit implementation is also vulnerable to this reduction attack as Eqn. (4) can be solved by the algorithm of Derbez et al. [17].

Moreover, an implicit implementation with affine encodings must also ensure that no solution (X, Y) with X diagonal and Y linear of Eqn. (3) can be obtained efficiently. In particular, if an adversary can obtain efficiently a random solution (X, Y), then the adversary can always try all (A, B) and all U until $X' = U^{-1} \circ A^{-1} \circ X$ is diagonal and $Y' = Y \circ B^{-1}$ affine; an encoded affine layer can easily be derived from (X', Y') afterwards. Thus, the number of affine self-equivalences (A, B) of T and the number of quasilinear-preserving graph automorphisms U that are not self-equivalences of T such be large enough to prevent an exhaustive search.

Our generic reduction subroutine provides insightful requirements for an implicit implementation with affine encodings, but it also leaves several open problems, such as estimating the complexity to obtain a solution of Eqn. (4) for a generic implicit implementation or analysing the case where the input encoding is non-linear. Our generic reduction did not cover either the case where additional countermeasures are added to the implicit implementation. In particular, the representation of the implicit round functions as systems of multivariate Boolean polynomials allows applying countermeasures from multivariate public-key cryptosystems (MPKC) such as adding extra variables [24], adding or removing equations [33] or adding perturbations [18]. Understanding the security provided by the implicit framework requires further research, but our generic reduction subroutine together with the algorithms that we mentioned to solve Eqns. (3) and (4) can be used as starting point.

In the next section we will describe the self-equivalences of the modular addition in order to later propose an implicit implementation of a cipher with modular addition as the non-linear layer.

5 Self-Equivalences of Modular Addition

In this section we describe a new method to compute self-equivalences of a function from a CCZ-equivalent function of low degree. We applied this method to the permuted modular addition $(x, x') \mapsto (x \boxplus x', x')$, obtaining for the first time the self-equivalences of this operation for wordsize $n \leq 64$. The permuted modular addition is frequently used in the non-linear layers of block ciphers, and we focus on this operation to propose in the next section an implicit implementation of a cipher using the permuted modular addition. However, our new method is of independent interest, and it can also be applied to the non-permuted modular addition or other functions CCZ-equivalent to low-degree functions.

5.1 Computing Self-Equivalences from a CCZ-Equivalent Function

CCZ-equivalence is an equivalence relation between functions based on their graphs. It was introduced in [12] by Carlet, Charpin and Zinoviev, and named after these authors.

Definition 5. *A function F is CCZ-equivalent to a function G if the graph of F can be transformed to the graph of G through an affine permutation, that is, if there exists an affine permutation $L_{G,F}$ such that $\mathit{\Gamma}_F = L_{G,F}(\mathit{\Gamma}_G)$.*

To obtain the self-equivalences of a high-degree function F that is CCZ-equivalent to a low-degree function G, the core of this method is to compute a subset of graph automorphisms of G by solving a system of low-degree equations. Our method exploits the relation between the graph automorphisms of two CCZ-equivalent functions and the functional equation that characterizes a graph automorphism, shown in the next two lemmas.

Lemma 4. *Let F and G be two CCZ-equivalent functions. Then, the graph automorphism groups of F and G are conjugates, that is, U is a graph automorphism of F if and only if $L_{G,F}^{-1} \circ U \circ L_{G,F}$ is a graph automorphism of G.*

Proof. If U is a graph automorphism of F, then

$$\mathit{\Gamma}_G = L_{G,F}^{-1}(\mathit{\Gamma}_F) = (L_{G,F}^{-1} \circ U)(\mathit{\Gamma}_F) = (L_{G,F}^{-1} \circ U \circ L_{G,F})(\mathit{\Gamma}_G)\,,$$

which proves the forward direction of the statement. The backward direction can be proven similarly. □

Lemma 5. *Let G be an n-bit function and let $U(x, y) = (U_1(x, y), U_2(x, y))$ be a $2n$-bit affine permutation. Then U is a graph automorphism of G if and only if U satisfies the functional equation*

$$U_2 \circ (\mathrm{Id}_n, G) = G \circ U_1 \circ (\mathrm{Id}_n, G)\,. \tag{5}$$

Proof. From Lemma 1, for any permutation U one has

$$U^{-1}(\mathit{\Gamma}_G) = \{(x, y) : U_2(x, y) = G(U_1(x, y))\}\,.$$

If U is a graph automorphism, then $\mathit{\Gamma}_G = U^{-1}(\mathit{\Gamma}_G)$. Therefore, $x = G(y)$ is equivalent to $U_2(x, y) = G(U_1(x, y))$. In particular, we have $U_2(x, G(x)) = G(U_1(x, G(x))$ for all x, which proves the forward direction.

If $U_2(x, G(x)) = G(U_1(x, G(x)))$ for all x, then $\mathit{\Gamma}_G = \{(x, y) : x = G(y)\}$ is contained in

$$\{(x, y) : U_2(x, y) = G(U_1(x, y))\} = U^{-1}(\mathit{\Gamma}_G)\,.$$

Since U^{-1} is a permutation, $\mathit{\Gamma}_G$ and $U^{-1}(\mathit{\Gamma}_G)$ have the same cardinality. Therefore, $\mathit{\Gamma}_G = U^{-1}(\mathit{\Gamma}_G)$, which implies that U is a graph automorphism of G. □

Our method consists of computing a set $\mathcal{U}$ of solutions of the functional equation given by Eqn. (5) with the additional constraints that U is invertible and that $U' = L_{G,F} \circ U \circ L_{G,F}^{-1}$ is diagonal (i.e., $U'(x,y) = (A(x), B(y))$). To reduce the degree of the invertibility constraint, we impose the invertibility constraint over the diagonal blocks of U' instead of over the entire U. After the set $\mathcal{U}$ is obtained, the self-equivalences of F can be extracted from the set $\mathcal{U}' = \{L_{G,F} \circ U \circ L_{G,F}^{-1} : U \in \mathcal{U}\}$ by applying Lemma 2, that is,

$$U'(x,y) = (A(x), B(y)) \in \mathcal{U}' \iff (A, B^{-1}) \text{ is a self-equivalence of } F\,.$$

A weaker variant of this method is to compute the affine right self-equivalences U of an implicit function P of G such that $U' = L_{G,F} \circ U \circ L_{G,F}^{-1}$ is diagonal. In this variant, Eqn. (5) is replaced by the functional equation $P = V \circ P \circ U$, and a third additional constraint is added to ensure that $V(0) = 0$ (Lemma 2). The advantage of this variant is that the degree of the functional equation is the same as the degree of the implicit function P, while the degree of Eqn. (5) is upper bounded by $2 \times \deg(G)$. However, not all the self-equivalences of F might be found if the weaker variant is used, since in general not all the graph automorphisms are right self-equivalences of an implicit function.

To obtain the set $\mathcal{U}$ in either of the two variants, we perform in practice the following three steps. First, we represent the functional equation and the additional constraints by a system of binary equations, perform Gaussian elimination on the system of equations, and then find an small set of solutions (e.g., 2^{20}) of the reduced system of equations with a SAT solver. Given the functional equation LHS = RHS, we build the system of equations by creating the vectorial Boolean function LHS $\oplus$ RHS as a system of multivariate polynomials, and then adding the equation $c = 0$ for each monomial coefficient c of LHS $\oplus$ RHS; the invertibility constraint is added to the system of equations using the determinant expression.

Second, we extract a set of candidate affine relations involving the monomial coefficients of U (seen as system of multivariate polynomials) from the small set of solutions previously obtained. For example, if the sum of two coefficients u_i and u_j is always 0 in all the solutions, we include the candidate affine relation $u_i + u_j = 0$. We call them *candidate* relations since they are satisfied for the small set of solutions but they might not be satisfied by the whole solution set. To find out whether a candidate relation $R = 0$ is satisfied by the whole solution set, we add the complementary relation $R = 1$ to the reduced system of equations previously obtained, and we check whether this new system is unsatisfiable with a SAT solver. If the system is unsatisfiable, then the relation $R = 0$ is satisfied by the whole solution set. We repeat this process for all candidate relations, obtaining a list of affine relations satisfied by the whole solution set.

Third, we build the final system of equations from the reduced system previously obtained by using the affine relations to fix coefficients of U and remove variables from the system. We then use a SAT solver to obtain a small number of solutions (e.g., 2^{20}) of the final system. Finally, we create the set $\mathcal{U}$ from these solutions and the affine relations, and we compute the cardinality of $\mathcal{U}$ by also taking into account the free coefficients of U not appearing in any of the equations of the final system nor in the affine relations. While further modifications and optimizations are possible, this practical approach is suitable for functional equations with sparse and quadratic central maps, such as the CCZ-equivalent function of the permuted modular addition.

Implementation. We have implemented this method in a new open-source library `BoolCrypt`[10]. Our library is written in Python and uses the library SageMath [39]. In particular, it relies on the PolyBori package of SageMath to create and manipulate binary equations, and on the SAT solver CryptoMiniSat [38] to find the solutions of systems of binary equations. Apart from this method, our library also provides many functionalities related to vectorial Boolean functions, self-equivalences and functional equations. We designed `BoolCrypt` with a modular structure and we documented it extensively, so that it can be useful and it can be adapted to future works.

5.2 Self-Equivalences and Graph Automorphisms of the Permuted Modular Addition

Let $\boxplus$ denote the $(2n, n)$-bit function representing the modular addition with wordsize n, i.e., $x \boxplus x' = x + x' \mod 2^n$. The modular addition has degree $n-1$, but Schulte-Geers proved that it is CCZ-equivalent to a quadratic function [36].

Theorem 1 ([36]). *Let $Q(x, x')$ be the $(2n, n)$-bit quadratic function given by*

$$Q(x, x') = (0, x_0 x'_0, x_0 x'_0 \oplus x_1 x'_1, \dots, x_0 x'_0 \oplus \cdots \oplus x_{n-2} x'_{n-2}) .$$

Then Q is CCZ-equivalent to $\boxplus$, and $\Gamma_{\boxplus} = L_{Q,\boxplus}(\Gamma_Q)$ where $L_{Q,\boxplus}$ is given by the block matrix

$$L_{Q,\boxplus} = \begin{pmatrix} \mathrm{Id}_n & 0_n & \mathrm{Id}_n \\ 0_n & \mathrm{Id}_n & \mathrm{Id}_n \\ \mathrm{Id}_n & \mathrm{Id}_n & \mathrm{Id}_n \end{pmatrix} .$$

Let Q' be the trivial implicit function of Q given by $Q'(x, x', y) = y \oplus Q(x, x')$. From Thm. 1 and Lemma 1, it is easy to see that

$$(Q' \circ L_{Q,\boxplus}^{-1})(x, x', y) = x \oplus x' \oplus y \oplus Q(x \oplus y, x' \oplus y) \tag{6}$$

is a quadratic implicit function of the modular addition.

The modular addition is mostly used in block ciphers in its permuted variant, $S(x, x') = (x \boxplus x', x')$, and this latter function is the focus of our work. The next lemma shows that a right self-equivalence of the permuted modular addition S is also a left self-equivalence up to the addition of some constants.

Lemma 6. *The permutation A is a right self-equivalence of S if and only if $\oplus_{(c_1, c_0)} \circ A \circ \oplus_{(c_1, c_0)}$ is a left self-equivalence of S, where c_0 and c_1 denote the n-bit values $(0, 0, \dots, 0)$ and $(1, 1, \dots, 1)$ respectively.*

[10] https://github.com/ranea/BoolCrypt

Proof. The inverse of the permuted modular addition is $S^{-1}(x, x') = (x \boxminus x', x')$, where $\boxminus = x - x' \mod 2^n$. Since the modular subtraction $\boxminus$ verifies $c_1 \oplus (x \boxminus y) = ((x \oplus c_1) \boxplus y)$ [22], we have the relation $S^{-1} = \oplus_{(c_1,c_0)} \circ S \circ \oplus_{(c_1,c_0)}$.

On the other hand, if (A, B) is a self-equivalence of S, then $S = B \circ S \circ A$, or equivalently $S^{-1} = A^{-1} \circ S^{-1} \circ B^{-1}$. By applying the previous relation and moving some terms, we have that

$$S = (\oplus_{(c_1,c_0)} \circ A^{-1} \circ \oplus_{(c_1,c_0)}) \circ S \circ (\oplus_{(c_1,c_0)} \circ B^{-1} \circ \oplus_{(c_1,c_0)}) \,.$$

Since the set of self-equivalences is a group, the inverse of $(\oplus_{(c_1,c_0)} \circ A^{-1} \circ \oplus_{(c_1,c_0)})$ is also a left self-equivalence of S, which proves the forward direction of the statement. The other direction can be proven similarly. $\square$

From Eqn. (6) and Thm. 1, it is easy to show that the $(2n, 2n)$-bit function $x, x' \mapsto (Q(x, x'), x')$ is CCZ-equivalent to the permuted modular addition S and that the following function is a quadratic implicit function of S:

$$T(x, x', y, y') = (x \oplus x' \oplus y \oplus Q(x \oplus y, x' \oplus y), x' \oplus y') \ . \tag{7}$$

Using the CCZ-equivalent quadratic function of the permuted[11] modular addition, we applied our method implemented in `BoolCrypt` to obtain the affine and linear self-equivalences for multiple wordsizes n. First, we obtained all the affine and linear self-equivalences for the wordsizes $n \in \{2, 3, 4, 5\}$, for which we could solve Eqn. (5) directly. For $n = 2$ we obtained 3×2^{10} affine and 3×2^8 linear self-equivalences, and for $n \in \{3, 4, 5\}$ we obtained $3 \times 2^{2n+8}$ affine and $3 \times 2^{2n+2}$ linear self-equivalences, respectively. For verification purposes, we also ran the affine and linear equivalence algorithms by Biryukov et al. [8], and we obtained the same number of linear and affine self-equivalences. In Appendix A and Appendix B we fully describe the affine and linear self-equivalence groups for $n = 4$.

For $n \geq 6$, we could not directly solve Eqn. (5). However, we noticed that the right (and by Lemma 6 also the left) affine self-equivalences for the small wordsizes have a common sparse matrix shape:

$$\left(\begin{array}{ccccc|ccccc}
\star & & & & & \star & & & & \\
\star & 1 & & & & \star & & & & \\
\vdots & & \ddots & & & \vdots & & & & \\
\star & & & 1 & & \star & & & & \\
\star & & & \star & 1 & \star & \star & \dots & \star & \star \\
\hline
\star & & & & & \star & & & & \\
\star & & & & & \star & 1 & & & \\
\vdots & & & & & \vdots & & \ddots & & \\
\star & & & & & \star & & & 1 & \\
\star & & & & & \star & & & \star & 1
\end{array}\right) ,$$

[11] The self-equivalence group of the permuted modular addition cannot be derived from that of the modular addition, as the permuted variant contains many non-diagonal self-equivalences.

where each of the four blocks denote an n-bit binary matrix, the $\star$ denotes a coefficient that can take the value 0 or 1 and empty spaces denote zero coefficients. In the constant vector of the affine self-equivalences we also noticed a common shape where one value b was repeated in all but 6 positions:

$$(\overbrace{\star\, b\, \cdots\, b \star \star}^{n} \,|\, \star\, b\, \cdots\, b \star \star) \ .$$

To make Eqn. (5) easier to solve, we pre-emptively fixed some coefficients of U to enforce that the resulting self-equivalences have the previous shape. Thus, we applied our method with wordsizes $n \in \{6, 7, \dots, 64\}$ and we obtained a subset of affine (resp. linear) self-equivalences with cardinality $3 \times 2^{2n+8}$ (resp. $3 \times 2^{2n+2}$). Note that for $n = 64$ this method takes only a few minutes to obtain these subsets of self-equivalences. Since the number $3 \times 2^{2n+8}$ (resp. $3 \times 2^{2n+2}$) is the cardinality of the whole affine (resp. linear) self-equivalence group for $n \in \{3, 4, 5\}$ and a lower bound of this cardinality for $6 \le n \le 64$, we conjecture that the total number of affine (resp. linear) self-equivalences for any wordsize $n \ge 3$ is $3 \times 2^{2n+8}$ (resp. $3 \times 2^{2n+2}$).

Using our method and our library `BoolCrypt`, we also obtained affine-quadratic self-equivalences (A, B) of the permuted modular addition, that is, with A affine and B quadratic. Affine-quadratic self-equivalences can be used for quadratic input encodings and affine output encodings of implicit implementations, as we will see in the next section. We first computed all affine-quadratic self-equivalences for the small wordsizes $n \in \{2, 4\}$. We obtained $19 \times 3^2 \times 2^{10} - 3 \times 2^{10}$ affine-quadratic self-equivalences for $n = 2$, and $3^2 \times 2^{3n+14} - 3 \times 2^{2n+8}$ for $n = 4$. Our method returned solutions including affine and affine-quadratic self-equivalences, and the factors 3×2^{10} and $3 \times 2^{2n+8}$ are the number of affine self-equivalences in the solutions for $n = 2$ and $n = 4$ respectively.

Affine-quadratic self-equivalences also share a common shape with several quadratic and linear terms that vanish. For example, the coefficients of the linear terms are of the form

$$\left(\begin{array}{ccccccc|ccccccc}
\star & \star & & & & & & \star & \star & & & & & \\
\star & \star & & & & & & \star & \star & & & & & \\
\star & \star & 1 & & & & & \star & \star & & & & & \\
\vdots & \vdots & & \ddots & & & & \vdots & \vdots & & & & & \\
\star & \star & & & 1 & & & \star & \star & & & & & \\
\star & \star & & & \star & 1 & & \star & \star & \star & \dots & \star & \star & \star \\
\star & \star & & & \star & \star & 1 & \star & \star & \star & \dots & \star & \star & \star \\
\hline
\star & \star & & & & & & \star & \star & & & & & \\
\star & \star & & & & & & \star & \star & 1 & & & & \\
\vdots & \vdots & & & & & & \vdots & \vdots & & \ddots & & & \\
\star & \star & & & & & & \star & \star & & & 1 & & \\
\star & \star & & & & & & \star & \star & \star & \cdots & \star & \star & \star \\
\star & \star & & & & & & \star & \star & \star & \cdots & \star & \star & \star
\end{array}\right) ,$$

and the constant vectors contain a repeated value b in all but 7 positions.

$$(\overbrace{\star\, b\, \cdots\, b \star \star}^{n} \,|\, \star\, b\, \cdots\, b \star \star \star) \ .$$

We used this shape to obtain a subset of affine-quadratic self-equivalences for the larger wordsizes $n \in \{5, 6, \dots, 64\}$ with cardinality $3^2 \times 2^{3n+14} - 3 \times 2^{2n+8}$. Thus, we conjecture that the number of affine-quadratic self-equivalences is equal or greater than $3^2 \times 2^{3n+14} - 3 \times 2^{2n+8}$ for any wordsize $n \geq 4$.

Finally, we also obtained quasilinear-preserving graph automorphisms U of the permuted modular addition by adapting the method of Sect. 5.1. In this case, the diagonal constraint is replaced by the quasilinear constraint over $T \circ U'$, where T is the implicit function defined in Eqn. (7). While in the original method the invertibility constraint is only applied to the diagonal blocks of U', to sample graph automorphisms the invertibility constraint needs to be applied to the entire U. Due to this more complex invertibility constraint, for small wordsizes $n \in \{2, 3, 4\}$ we could find some but not all the quasilinear-preserving graph automorphisms. For large wordsizes $n \in \{5, 6, \dots, 64\}$, we could sample quasilinear-preserving graph automorphisms by finding some solutions without the invertibility constraint and with some coefficients pre-emptively fixed, and then searching among the solutions for an invertible function. In our experiments, invertible solutions were quickly found after checking few solutions.

We noticed that if the invertibility constraint is not included in the system of equations, many coefficients of U do not appear in any equation. For example, for the wordsizes 3, 4, 5, 6, 7 and 8, we observed $54, 80, 110, 144, 182$ and 244 free coefficients, respectively, and we found at least 2^{20} solutions for each possible assignment of the free coefficients, that is, $2^{74}, 2^{100}, 2^{130}, 2^{174}, 2^{212}$ and 2^{274} solutions respectively. Moreover, all the quasilinear-preserving graph automorphisms that we sampled in our experiments were not right self-equivalences of T. Thus, we conjecture that the number of quasilinear-preserving graph automorphisms that are not right self-equivalences of T is exponential in the input size of the permuted modular addition.

As a summary, using the method described in Sect. 5.1 for the permuted modular addition with wordsize $4 \leq n \leq 64$, we found affine, linear, and affine-quadratic self-equivalence subsets with cardinality $3 \times 2^{2n+8}$, $3 \times 2^{2n+2}$ and $3^2 \times 2^{3n+14} - 3 \times 2^{2n+8}$, respectively. We could not obtain a lower bound on the number of quasilinear-preserving graph automorphisms that are not right self-equivalences of T, but using our method we can sample this type of graph automorphisms for wordsizes $n \leq 64$.

The motivation behind these self-equivalences and graph automorphisms is to propose in the next section an implicit implementation of an arbitrary cipher with the permuted modular addition as the non-linear layer. Since the affine, linear, and affine-quadratic self-equivalence subsets are large enough, we did not focus on finding all the self-equivalences and we leave it as an open problem. On the other hand, finding a lower bound on the number of quasilinear-preserving graph automorphisms that are not right self-equivalences of T would have been useful for the security analysis of the implicit implementation, but we were not able to obtain this bound and we leave it for future work.

6 An Implicit Implementation of an ARX Cipher

In this section we describe how to build an implementation of a generic iterated block cipher with an arbitrary affine layer and with the permuted modular addition as the non-linear layer. Most Addition-Rotation-XOR (ARX) ciphers fall under this description, and our method is the first white-box design that can be applied to ARX ciphers.

Let $E_k = E^{(n_r)} \circ E^{(n_r-1)} \circ \cdots \circ E^{(1)}$ be the encryption function of an n-bit iterated block cipher (n even), and let $E^{(i)} = AL^{(i)} \circ S$ be its ith round function where S is the permuted modular addition[12] with wordsize $n/2$ (i.e., input bitsize n) and $AL^{(i)}$ is an arbitrary n-bit affine layer containing the i-th round key $k^{(i)}$.

To build the quasilinear implicit function $P^{(i)}$ of the ith round, we use the approach described in Sect. 3.1 which defines $P^{(i)}$ as

$$P^{(i)} = V^{(i)} \circ T \circ U^{(i)} \circ \left(\mathrm{Id}_n, (AL^{(i)})^{-1}\right) \circ \left(I^{(i)}, (O^{(i)})^{-1}\right),$$

where $V^{(i)}$ is a linear permutation, T is the quasilinear implicit function of S, $U^{(i)}$ is a quasilinear-preserving affine graph automorphism of S that does not belong to the right self-equivalences of T, and $(I^{(i)}, O^{(i)})$ are the round encodings.

In this case, $T(x, x', y, y')$ is the quadratic implicit function of the permuted modular addition, previously defined in Eqn. (7). It is easy to see that T is quasilinear, as all its quadratic monomials contain an input variable (from the variable vectors x or x'). For $V^{(i)}$ we sample a random linear permutation, and we sample a random $U^{(i)}$ with the method described in Sect. 5.2.

For the round encodings, we sample an affine permutation $C^{(i+1)}$ and an affine-quadratic self-equivalence $(A^{(i)}, B^{(i)})$ of $E^{(i)}$, and we build the round encodings as

$$(I^{(i)}, O^{(i)}) = \left(A^{(i)} \circ B^{(i-1)} \circ (C^{(i)})^{-1},\ C^{(i+1)}\right). \tag{8}$$

In the first input round encoding (i.e., the external input encoding), for $B^{(0)}$ we sample a random quadratic permutation.

Since the self-equivalence groups of S and $E^{(i)}$ are conjugates, that is,

$$\begin{aligned}\left(A^{(i)}, B^{(i)}\right) \text{ is a self-equivalence of } S \iff \\ \left(A^{(i)}, AL^{(i)} \circ B^{(i)} \circ (AL^{(i)})^{-1}\right) \text{ is a self-equivalence of } E^{(i)},\end{aligned}$$

we can sample an affine-quadratic self-equivalence of $E^{(i)}$ from the subset of affine-quadratic self-equivalences of the permuted modular addition obtained

[12] For simplicity we restrict the n-bit non-linear layer to contain a single permuted modular addition with wordsize $n/2$, but our method can easily be extended to non-linear layers composed of smaller permuted modular additions.

in Sect. 5.2. In this construction, the pairs $(A^{(i)}, B^{(i)})$ can also be taken as the identity functions, and the choice of using affine-quadratic self-equivalences leads to a trade-off between security and performance that we will discuss later.

Note that the round encodings do not satisfy the cancellation rule $I^{(i+1)} \circ O^{(i)} = \mathrm{Id}_n$ from Definition 2 but the cancellation rule given by Eqn. (1), since

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

Figure 2 depicts how the round encodings are cancelled in two consecutive rounds. The round encodings could also be defined as

$$\left(I^{(i)}, O^{(i)}\right) = \left(B^{(i-1)} \circ (C^{(i)})^{-1}, C^{(i+1)} \circ (B^{(i)})^{-1}\right), \tag{9}$$

which satisfy the cancellation rule $I^{(i+1)} \circ O^{(i)} = \mathrm{Id}_n$. While both Eqn. (8) and Eqn. (9) define the same encoded round function, we will use Eqn. (8) where the output round encoding is affine to preserve the quasilinear property (see Sect. 3.1).

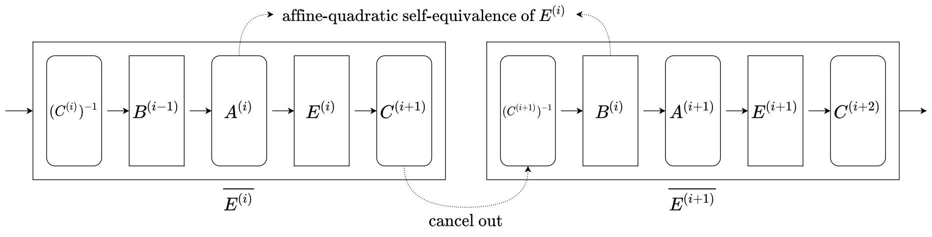

Fig. 2. Two consecutive encoded rounds, $\overline{E^{(i)}}$ and $\overline{E^{(i+1)}}$, with the round encodings defined by Eqn. (8). Rounded blocks denote affine functions and rectangular blocks denote non-linear functions.

As a result, we have built an implicit implementation of E_k given by the quasilinear implicit functions $\{P^{(1)}, P^{(2)}, \ldots, P^{(n_r)}\}$. Let $\overline{E_k}$ be the underlying encoded implementation, whose round encodings are given by Eqn. (8). Note that $\overline{E_k}$ is not functionally equivalent to E_k since

$$\overline{E_k} = \left(C^{(n_r+1)} \circ (B^{(n_r)})^{-1}\right) \circ E_k \circ \left(B^{(0)} \circ (C^{(1)})^{-1}\right).$$

Thus, the implicit implementation is not functionally equivalent to E_k either.

The degree of the implicit round functions $P^{(i)}$ depends on whether affine-quadratic self-equivalences are used in the round encodings. If they are not used, the functions $P^{(i)}$ are quadratic. Otherwise, the functions $P^{(i)}$ are quartic, or cubic if the affine-quadratic self-equivalences are chosen carefully. Higher degree self-equivalences (e.g., affine-cubic, affine-quartic, etc.) could also be used at the cost of increasing the degree of the implicit round functions.

Table 1 shows an upper bound on the memory required by an implicit round function for different degrees and bitsizes. As shown in the table, implicit implementations with bitsize $n = 64$ are quite practical; for bitsize $n = 128$ implicit implementations are practical with quadratic or cubic rounds.

Table 1. Upper bound on the size in megabytes (MB) of a $(2n, n)$-bit implicit round function $P^{(i)}$ for different degrees and bitsizes n.

Cipher blocksize n	Degree of $P^{(i)}$	Size of $P^{(i)}$
32	2	0.01 MB
32	3	0.09 MB
32	4	0.23 MB
64	2	0.05 MB
64	3	1.42 MB
64	4	6.50 MB
128	2	0.40 MB
128	3	22.50 MB
128	4	193.19 MB

Note that an implementation with multivariate binary polynomials of the underlying encoded implementation would be infeasible (even without affine-quadratic self-equivalences) due to its size, since the permuted modular addition has degree n. A CEJO implementation of the underlying encoded implementation would also have impractical size as the non-linear layer cannot be written as the composition of smaller functions. While an implicit implementation introduces a significant overhead in the running time and a severe overhead in the size as shown in Table 1, the implicit framework is the first method that provides practical white-box implementations of ARX ciphers.

Security Analysis. Let $\mathcal{I}$ be an implicit implementation following the method described in this section, and let E_k be the underlying block cipher with the permuted modular addition as the non-linear operation. In Sect. 4.1 we showed that implicit implementations that use non-linear encodings or where the non-linear layer of the cipher is not composed of small S-boxes are secure against all known generic key-extraction attacks. Therefore, if $\mathcal{I}$ uses quadratic input encodings (i.e., affine-quadratic self-equivalences) or the non-linear layer of E_k is not composed of small permuted modular additions, then known generic key-extraction attacks cannot extract the key from $\mathcal{I}$. Recall that a self-equivalence implementation of E_k has the structural weakness mentioned in Sect. 4.1, but $\mathcal{I}$ does not exhibit this weakness due to the large affine permutations C^i in the round encodings.

For the rest of the analysis, we will restrict $\mathcal{I}$ to have affine round encodings and E_k to have a permuted modular addition with wordsize $n/2$ as the non-linear layer. In Sect. 4.2 we also argued that an implicit implementation with affine encodings must ensure that no solution of Eqn. (4) and no diagonal-linear solution of Eqn. (3) can be obtained efficiently.

The central map in Eqn. (4) is the permuted modular addition, a triangular function with degree n. If an attacker would have access to the coefficients of Eqn. (4), the attacker could efficiently obtain a solution using the structural attack by Wolf et al. [42]. This structural attack can solve in polynomial time a functional equation with affine permutations as unknowns and with a non-linear (not necessarily quadratic) triangular central map. However, the attack by Wolf et al. assumes that the coefficients of the functional equation are public. This is not the case since an attacker only has black-box access to Eqn. (4), that is, the attacker can only evaluate Eqn. (4) by using the implicit round functions.

Regarding Eqn. (3), using the method of Sect. 5.1 we obtained that $3^n \times 2^{11n-4}$ is a lower bound on the cardinality of the affine self-equivalence group of T for wordsizes $2 \leq n \leq 64$. While we were not able to obtain a lower bound on the number of quasilinear-preserving graph automorphisms U that are not right self-equivalences of T, the large number of affine self-equivalences (A, B) of T prevents an attacker from trying all (A, B) and U to transform a random solution (X, Y) into (X', Y') with X diagonal and Y linear.

To conclude, an implicit implementation following the method described in this section prevents all known generic key-extraction attacks if quadratic input encodings are used or the non-linear layer of the underlying cipher is given by a large permuted modular addition. Our new generic attack to the implicit framework based on Eqns. (3) and (4) was also not able to break an implicit implementation with affine encodings. This contribution also leaves several open problems such as the cost of an attack dealing with quadratic input encodings, whether it is possible to extend the structural attack by Wolf et al. to only black-box access or even finding new algorithms to solve Eqn. (3) or to obtain a diagonal-linear solution of Eqn. (4) without trying all (A, B) and U.

Implementation. As mentioned at the beginning of Sect. 4.1, the implicit framework is a novel design which requires further research on new attacks. Rather than proposing a particular implicit implementation of some ARX cipher, we provide instead an open-source Python script[13] that generates an implicit implementation of any given ARX cipher following the method described in Sect. 6.

Our script can be used for any cipher with the permuted modular addition as the non-linear layer. Given as input the affine layer of each round, the script generates the implicit implementation in an automated way. As an example, we have included an additional script that generates the affine layers of `SPECK` [5], but other ciphers based on the permuted modular addition can be easily added.

Moreover, the script provides many options that can be enabled or disabled, such as the affine-quadratic self-equivalences, the external encodings, an additional countermeasure based on Bringer et al.'s perturbations [11] or whether to export the implicit implementation to a Python function or to a C source file.

[13] https://github.com/ranea/whiteboxarx

This script can generate many variants of implicit implementations of any ARX cipher, and we hope that these many practical examples encourage further analysis of implicit implementations.

7 Conclusion

Despite the total lack of secure candidates of white-box implementations, many recent works have focused on attack techniques, but only a few on design strategies. That is why in this work we addressed the challenging problem of designing white-box implementations of existing ciphers by proposing implicit implementations. On top of that, we proposed a generic method to obtain self-equivalences and graph automorphisms, which can be of independent interest together with `BoolCrypt`.

While we analysed the resistance of implicit implementations against the existing generic key-extraction attacks, the implicit framework is a new design that is very different from previous CEJO or self-equivalence implementations; many open problems arise when analysing new attacks. Understanding the security of implicit implementations requires further research, and we encourage the cryptographic community to participate by providing many practical examples with our script.

On the other hand, in this work we focused on implicit implementations with external encodings and with the permuted modular addition as the non-linear layer of the cipher, but future work could explore other non-linear layers such as large power functions or the security of implicit implementations without external encodings against automated white-box attacks based on grey-box techniques.

Acknowledgements. Adrián Ranea is supported by a PhD Fellowship from the Research Foundation – Flanders (FWO). The authors would like to thank the anonymous reviewers for their comments and suggestions.

References

1. Albrecht, M.R., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In: ASIACRYPT (1). Lecture Notes in Computer Science, vol. 10031, pp. 191–219 (2016)
2. Amadori, A., Michiels, W., Roelse, P.: A DFA attack on white-box implementations of AES with external encodings. In: SAC. Lecture Notes in Computer Science, vol. 11959, pp. 591–617. Springer (2019)
3. Baek, C.H., Cheon, J.H., Hong, H.: White-box AES implementation revisited. J. Commun. Networks **18**(3), 273–287 (2016)
4. Banik, S., Bogdanov, A., Isobe, T., Jepsen, M.B.: Analysis of software countermeasures for whitebox encryption. IACR Trans. Symmetric Cryptol. **2017**(1), 307–328 (2017)

5. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK families of lightweight block ciphers. IACR Cryptol. ePrint Arch. p. 404 (2013)
6. Biryukov, A., Bouillaguet, C., Khovratovich, D.: Cryptographic schemes based on the ASASA structure: Black-box, white-box, and public-key (extended abstract). In: ASIACRYPT (1). Lecture Notes in Computer Science, vol. 8873, pp. 63–84. Springer (2014)
7. Biryukov, A., De Cannière, C.: Block ciphers and systems of quadratic equations. In: FSE. Lecture Notes in Computer Science, vol. 2887, pp. 274–289. Springer (2003)
8. Biryukov, A., De Cannière, C., Braeken, A., Preneel, B.: A toolbox for cryptanalysis: Linear and affine equivalence algorithms. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 2656, pp. 33–50. Springer (2003)
9. Biryukov, A., Udovenko, A.: Attacks and countermeasures for white-box designs. In: ASIACRYPT (2). Lecture Notes in Computer Science, vol. 11273, pp. 373–402. Springer (2018)
10. Bos, J.W., Hubain, C., Michiels, W., Teuwen, P.: Differential computation analysis: Hiding your white-box designs is not enough. In: CHES. Lecture Notes in Computer Science, vol. 9813, pp. 215–236. Springer (2016)
11. Bringer, J., Chabanne, H., Dottax, E.: White box cryptography: Another attempt. IACR Cryptology ePrint Archive **2006**, 468 (2006)
12. Carlet, C., Charpin, P., Zinoviev, V.A.: Codes, bent functions and permutations suitable for des-like cryptosystems. Des. Codes Cryptogr. **15**(2), 125–156 (1998)
13. Chow, S., Eisen, P.A., Johnson, H., van Oorschot, P.C.: White-box cryptography and an AES implementation. In: Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 2595, pp. 250–270. Springer (2002)
14. Chow, S., Eisen, P.A., Johnson, H., van Oorschot, P.C.: A white-box DES implementation for DRM applications. In: Digital Rights Management Workshop. Lecture Notes in Computer Science, vol. 2696, pp. 1–15. Springer (2002)
15. De Cannière, C.: Analysis and Design of Symmetric Encryption Algorithms. Ph.D. thesis, Katholieke Universiteit Leuven (2007), bart Preneel (promotor)
16. Delerablée, C., Lepoint, T., Paillier, P., Rivain, M.: White-box security notions for symmetric encryption schemes. In: Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 8282, pp. 247–264. Springer (2013)
17. Derbez, P., Fouque, P., Lambin, B., Minaud, B.: On recovering affine encodings in white-box implementations. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(3), 121–149 (2018)
18. Ding, J.: A new variant of the matsumoto-imai cryptosystem through perturbation. In: Public Key Cryptography. Lecture Notes in Computer Science, vol. 2947, pp. 305–318. Springer (2004)
19. Faugère, J., Perret, L.: Polynomial equivalence problems: Algorithmic and theoretical aspects. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 4004, pp. 30–47. Springer (2006)
20. Faugère, J., Perret, L.: An efficient algorithm for decomposing multivariate polynomials and its applications to cryptography. J. Symb. Comput. **44**(12), 1676–1689 (2009)
21. Gupta, K.C., Ray, I.G.: Finding biaffine and quadratic equations for s-boxes based on power mappings. IEEE Trans. Inf. Theory **61**(4), 2200–2209 (2015)
22. Henry S. Warren, J.: Hacker's delight. Addison-Wesley (2003)
23. Karroumi, M.: Protecting white-box AES with dual ciphers. In: ICISC. Lecture Notes in Computer Science, vol. 6829, pp. 278–291. Springer (2010)

24. Kipnis, A., Patarin, J., Goubin, L.: Unbalanced oil and vinegar signature schemes. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 1592, pp. 206–222. Springer (1999)
25. Lin, D., Faugère, J., Perret, L., Wang, T.: On enumeration of polynomial equivalence classes and their application to MPKC. Finite Fields and Their Applications **18**(2), 283–302 (2012)
26. Link, H.E., Neumann, W.D.: Clarifying obfuscation: Improving the security of white-box DES. In: ITCC (1). pp. 679–684. IEEE Computer Society (2005)
27. Luo, R., Lai, X., You, R.: A new attempt of white-box AES implementation. In: SPAC. pp. 423–429. IEEE (2014)
28. McMillion, B., Sullivan, N.: Attacking white-box AES constructions. In: SPRO@CCS. pp. 85–90. ACM (2016)
29. Michiels, W., Gorissen, P., Hollmann, H.D.L.: Cryptanalysis of a generic class of white-box implementations. In: Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 5381, pp. 414–428. Springer (2008)
30. Minaud, B., Derbez, P., Fouque, P., Karpman, P.: Key-recovery attacks on ASASA. J. Cryptol. **31**(3), 845–884 (2018)
31. Patarin, J.: Hidden fields equations (HFE) and isomorphisms of polynomials (IP): two new families of asymmetric algorithms. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 1070, pp. 33–48. Springer (1996)
32. Patarin, J., Goubin, L.: Asymmetric cryptography with s-boxes. In: ICICS. Lecture Notes in Computer Science, vol. 1334, pp. 369–380. Springer (1997)
33. Patarin, J., Goubin, L., Courtois, N.T.: $C^{*}{}_{-+}$ and HM: variations around two schemes of t. matsumoto and h. imai. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 1514, pp. 35–49. Springer (1998)
34. Ranea, A., Preneel, B.: On self-equivalence encodings in white-box implementations. In: SAC. Lecture Notes in Computer Science, vol. 12804, pp. 639–669. Springer (2020)
35. Rivain, M., Wang, J.: Analysis and improvement of differential computation attacks against internally-encoded white-box implementations. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(2), 225–255 (2019)
36. Schulte-Geers, E.: On ccz-equivalence of addition mod 2^n. Des. Codes Cryptogr. **66**(1-3), 111–127 (2013)
37. Shi, Y., Wei, W., He, Z.: A lightweight white-box symmetric encryption algorithm against node capture for wsns. Sensors **15**(5), 11928–11952 (2015)
38. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: SAT. Lecture Notes in Computer Science, vol. 5584, pp. 244–257. Springer (2009)
39. The Sage Developers: SageMath, the Sage Mathematics Software System (Version 9.1) (2021), `https://www.sagemath.org`
40. Capture the Flag Challenge - The WhibOx Contest (2007), https://whibox.io/contests/2017/
41. Wiemers, A.: A note on invariant linear transformations in multivariate public key cryptography. IACR Cryptol. ePrint Arch. p. 602 (2012)
42. Wolf, C., Braeken, A., Preneel, B.: On the security of stepwise triangular systems. Des. Codes Cryptogr. **40**(3), 285–302 (2006)
43. Xiao, Y., Lai, X.: A secure implementation of white-box aes. In: 2009 2nd International Conference on Computer Science and its Applications. pp. 1–6. IEEE (2009)

A Affine Self-Equivalences of the Permuted Modular Addition with Wordsize 4

Let $L(A)$ and $C(A)$ be the linear part and the constant vector, respectively, of an affine function A. Any affine self-equivalence (A, B) of the 8-bit permuted modular addition (wordsize 4) is of the form

$$L(A) = \begin{pmatrix} c_1+c_8+c_9 & 0 & 0 & 0 & c_1+c_8 & 0 & 0 & 0 \\ d_0+d_2 & 1 & 0 & 0 & d_1 & 0 & 0 & 0 \\ d_0+d_2 & 0 & 1 & 0 & d_1 & 0 & 0 & 0 \\ d_1+c_6+c_{12}+c_{15} & 0 & c_4+c_7 & 1 & d_1+c_6+c_{12} & c_{13} & c_4+c_7+c_{16} & c_5+1 \\ c_0+c_8+c_9 & 0 & 0 & 0 & c_0+c_1+c_8 & 0 & 0 & 0 \\ d_0+d_2 & 0 & 0 & 0 & d_1 & 1 & 0 & 0 \\ d_0+d_2 & 0 & 0 & 0 & d_1 & 0 & 1 & 0 \\ d_1+c_2+c_{12}+c_{15} & 0 & 0 & 0 & d_1+c_2+c_6+c_{12} & c_3+c_{13} & c_7+c_{16} & 1 \end{pmatrix}$$

$$C(A) = (c_{10}+c_{11},\ c_{10}c_{11}+c_{11},\ c_4+d_3,\ c_{17}+c_{18},\ c_{11},\ c_{10}c_{11}+c_{11},\ d_3+c_{14}+c_{16},\ d_4)^T$$

$$L(B^{-1}) = \begin{pmatrix} c_0+c_1 & 0 & 0 & 0 & c_1 & 0 & 0 & 0 \\ d_0+d_2 & 1 & 0 & 0 & d_2+c_1c_{10} & 0 & 0 & 0 \\ d_0+d_2 & 0 & 1 & 0 & d_2+c_1c_{10} & 0 & 0 & 0 \\ d_0+c_2+c_6 & 0 & c_4+c_{14}+c_{16} & 1 & c_1c_{10}+c_6 & c_3 & d_5+c_7 & c_5+1 \\ c_0+c_8+c_9 & 0 & 0 & 0 & c_1+c_9 & 0 & 0 & 0 \\ d_0+d_2 & 0 & 0 & 0 & d_2+c_1c_{10} & 1 & 0 & 0 \\ d_0+d_2 & 0 & 0 & 0 & d_2+c_1c_{10} & 0 & 1 & 0 \\ d_1+c_2+c_{12}+c_{15} & 0 & 0 & 0 & c_6+c_{15} & c_3+c_{13} & c_7+c_{16} & 1 \end{pmatrix}$$

$$C(B^{-1}) = (c_{10},\ c_{10}c_{11}+c_{11},\ c_{10}c_{11}+c_{11}+d_5,\ c_{17},\ c_{11},\ c_{10}c_{11}+c_{11},\ d_3+c_{14}+c_{16},\ d_4)^T$$

where the binary coefficients c_i and d_j satisfy the following constraints

$$\begin{aligned} 0 &= d_0 + c_0c_1 + c_0c_8 + c_0c_{10} + c_0c_{11} + c_1 + c_8c_{10} + c_8 \\ 0 &= d_1 + d_0 + c_1c_{10} \\ 0 &= d_2 + c_1c_9 + c_1c_{11} + c_9c_{10} + c_9 + 1 \\ 0 &= d_3 + c_7 + c_{10}c_{11} + c_{11} \\ 0 &= d_4 + c_4c_7 + c_4c_{14} + c_4c_{16} + c_7c_{14} + c_7c_{16} + d_3 + c_{18} \\ 0 &= d_5 + c_4 + c_{14} + c_{16} \\ 0 &= c_0c_9 + c_1c_8 + c_1 + 1\,. \end{aligned}$$

The coefficients d_j are just short labels to denote large expressions involving coefficients c_i. Among the 19 c_i coefficients, 15 are free variables and (c_0, c_1, c_8, c_9) are restricted by the constraint $c_0c_9 + c_1c_8 + c_1 + 1$. This constraint excludes 10 out of the 2^4 assignments of (c_0, c_1, c_8, c_9). Therefore, the number of affine self-equivalences is $2^{15} \times (2^4 - 10) = 196\,608$, which corresponds to $3 \times 2^{2n+8}$ for $n = 4$.

B Linear Self-Equivalences of the Permuted Modular Addition with Wordsize 4

Any linear self-equivalence (A, B) of the 8-bit permuted modular addition (wordsize 4) is of the form

$$A = \begin{pmatrix} d_1 + c_8 & 0 & 0 & 0 & d_1 & 0 & 0 & 0 \\ d_3 & 1 & 0 & 0 & d_0 + d_1 & 0 & 0 & 0 \\ d_3 & 0 & 1 & 0 & d_0 + d_1 & 0 & 0 & 0 \\ d_4 + c_6 + c_{12} & 0 & 0 & 1 & d_4 + c_6 & c_{10} & c_{11} & c_5 + 1 \\ c_0 + c_7 + c_8 & 0 & 0 & 0 & c_0 + d_1 & 0 & 0 & 0 \\ d_3 & 0 & 0 & 0 & d_0 + d_1 & 1 & 0 & 0 \\ d_3 & 0 & 0 & 0 & d_0 + d_1 & 0 & 1 & 0 \\ d_4 + c_2 + c_{12} & 0 & 0 & 0 & d_4 + c_2 + c_6 & c_3 + c_{10} & c_4 + c_{11} & 1 \end{pmatrix}$$

$$B^{-1} = \begin{pmatrix} c_0 + c_1 & 0 & 0 & 0 & c_1 & 0 & 0 & 0 \\ d_3 & 1 & 0 & 0 & d_2 & 0 & 0 & 0 \\ d_3 & 0 & 1 & 0 & d_2 & 0 & 0 & 0 \\ d_3 + c_2 + c_6 & 0 & 0 & 1 & d_2 + c_6 & c_3 & c_4 & c_5 + 1 \\ c_0 + c_7 + c_8 & 0 & 0 & 0 & c_1 + c_8 & 0 & 0 & 0 \\ d_3 & 0 & 0 & 0 & d_2 & 1 & 0 & 0 \\ d_3 & 0 & 0 & 0 & d_2 & 0 & 1 & 0 \\ d_4 + c_2 + c_{12} & 0 & 0 & 0 & c_6 + c_{12} & c_3 + c_{10} & c_4 + c_{11} & 1 \end{pmatrix}$$

where the binary coefficients c_i and d_j satisfy the following constraints

$$\begin{aligned} 0 &= d_0 + c_0 c_1 + c_0 c_7 \\ 0 &= d_1 + c_1 + c_7 \\ 0 &= d_2 + c_1 c_8 + c_8 + 1 \\ 0 &= d_3 + d_0 + d_1 + d_2 \\ 0 &= d_4 + d_0 + d_1 + c_9 \\ 0 &= c_0 c_8 + c_1 c_7 + c_1 + 1\,. \end{aligned}$$

The coefficients d_j are simply short labels to denote large expressions involving coefficients c_i. Among the 13 c_i coefficients, 9 are free variables and (c_0, c_1, c_7, c_8) are restricted by the constraint $c_0 c_8 + c_1 c_7 + c_1 + 1$. This constraint excludes 10 out of the 2^4 assignments of (c_0, c_1, c_7, c_8). Therefore, the number of linear self-equivalences is $2^9 \times (2^4 - 10) = 3072$, which corresponds to $3 \times 2^{2n+2}$ for $n = 4$.