Applicable Algebra in Engineering, Communication and Computing
https://doi.org/10.1007/s00200-023-00602-w

ORIGINAL PAPER

Fourteen years of cube attacks

Marco Cianfriglia[1] · Elia Onofri[1] · Silvia Onofri[2] · Marco Pedicini[1]

Received: 5 May 2022 / Revised: 12 March 2023 / Accepted: 12 March 2023

Abstract
Algebraic Cryptanalysis is a widely used technique that tackles the problem of breaking ciphers mainly relying on the ability to express a cryptosystem as a solvable polynomial system. Each output bit/word can be expressed as a polynomial equation in the cipher's inputs—namely the key and the plaintext or the initialisation vector bits/words. A part of research in this area consists in finding suitable algebraic structures where polynomial systems can be effectively solved, e.g., by computing Gröbner bases. In 2009, Dinur and Shamir proposed the *cube attack*, a chosen plaintext algebraic cryptanalysis technique for the offline acquisition of an equivalent system by means of monomial reduction; interpolation on *cubes* in the space of variables enables retrieving a linear polynomial system, hence making it exploitable in the online phase to recover the secret key. Since its introduction, this attack has received both many criticisms and endorsements from the crypto community; this work aims at providing, under a unified notation, a complete state-of-the-art review of recent developments by categorising contributions in five classes. We conclude the work with an in-depth description of the *kite attack framework*, a cipher-independent tool that implements cube attacks on GPUs. Mickey2.0 is adopted as a showcase.

Keywords Cryptanalysis · Cube attacks · Kite attack · Algebraic attacks · GPU implementation · Mickey2.0

Mathematics Subject Classification 94A60 · 11T71

The first author has been partially supported by Institute for Applied Computing 'Mauro Picone' IAC – CNR.

Elia Onofri
eonofri@uniroma3.it

Extended author information available on the last page of the article

1 Introduction

In modern cryptographic algorithms, security assumptions which follow the *computational model* are usually based on problems whose difficulty is provable.

Extensively used in many cryptographic applications, among the mathematical problems known to be difficult, a central question concerns how to find the solution to a system of polynomial equations. Solving large systems of multivariate polynomial equations reduces to a more general problem: how to represent in a "standard" way a (multivariable) polynomial ideal (see [23]).

The problem is simple if the degree of polynomials is one since essential algebraic tools (like Gaussian reduction) provide feasible resolution algorithms. However complexity rises quickly when degrees are higher than one; in fact, the problem becomes NP-complete also under basic assumptions, like quadratic equations in $\mathbb{F}_2$ only.

On the other hand, it is also true that any algorithm may be seen, extensionally, as a black box computing a boolean function. This is *a fortiori* true for cryptographic algorithms where outputs can be sketched as the bits generated by inputs evaluation regardless of the intrinsic structure of the algorithm itself. Moreover, under the coercion of domain and codomain having the algebraic structure of a finite field, such a representation of the algorithm can be built from the evaluation of enough points in the domain. This analysis paves the way for a wide variety of cryptanalysis techniques based on the reformulation of a cryptosystem as a polynomial function over $\mathbb{F}_2$.

Many efforts were devoted in recent years to find efficient techniques to solve high-degree multivariate polynomial systems (see [71]). We annoverate Gröbner bases (see [5, 41, 70]) and linearisation techniques (see XL in [24] and XSL in [25]) as some of the most promising approaches actually fading after the same fate: Gröbner bases, in particular, despite being a very general and versatile solution, are unfeasible in many practical cases due to their computational cost.

Such failures led to the objective of finding useful algebraic relations between cryptographic schemes' input and output as a research topic meagre in results for a number of years.

In 2009, however, everything changed as a novel approach introduced by Dinur and Shamir at Eurocrypt'09 brought nourishment to algebraic cryptanalysis research branch. In [31], the first paper of a long run, authors introduced the *cube attack*, a fresh technique employed to extract a linear system of equations from the polynomial specification of a given cryptosystem. In a few years, such a technique diversified to become a big family of attacks, consisting of many variants including property testers [7] and differential trails [112] amongst the others.

The newly born family was characterised by the just introduced concept of analysing a cypher as a black-box tweakable polynomial, a polynomial where some variables could be set at will during the attack; the target of the attack consists, in fact, in building multiple linear equations and collect them in a system that is easy-to-solve. We feel comfortable admitting that the valuable original idea here exploited is not the effort from the sole Dinur and Shamir. They surely have the merit of clearly pointing out crucial steps and making organised this technique in all its aspects; higher-order differential cryptanalysis mentioned by Lai [59] and Knudsen [58] in the 1990s probably contributed to its successful development, as well as Vielhaber's AIDA (algebraic IV differential attacks) [104].

The scope of this paper is to give a wide view of the complex and entangled development of the cube attack family, by unravelling the various contributions and presenting them in a more cohesive view. The work is organised as a survey of the many concepts developed around the cube attack. Nevertheless, we work on all the examples of the application of the cube attack, to uniform to a common notation we introduced in [77].

In Sect. 2, we revise and extend our novel notation, already introduced in [77], and we revise the original cube attack applied on the binary field $\mathbb{F}_2$. Then we extend it to a general finite field $\mathbb{F}_q$ and we provide some consideration on cube search.

Section 3 provides a summary of various techniques that fall in the macro-family of cube attacks, here divided into five main categories.

In Sect. 4, we revise the few attempts of implementing cube attack techniques, while we provide it with a Sect. 5, where we give a list of the best-known attacks on real-world ciphers due to the various families techniques.

Finally, we conclude this work with some considerations in Sect. 6 and we provide in "Appendix A" a spot "dive–in" into one of the most recent frameworks for cube attacks, implementing the *kite attack*.

2 The cube attack

2.1 Notation

Any field of research has its own nomenclature and cryptography makes no exception. However, cube attacks in particular never saw a common agreement on how to refer to its various key concepts. For this reason, each research line developed its own language and notation, often incoherent with the others. Do mind, as a simple example, that the name "Conditional Cube Attacks" is overloaded in literature: usually, it refers to an extension of "Cube Testers", however, it is used also in relation to cube attacks where conditions are imposed a priori.

Diversity in notations makes it difficult to agree upon the specific novel contributions brought by each research; it is not unusual that concepts are claimed as novel and revolutionary, while simply being reformulations of already-known results.

We found this as a valid motivation to propose a novel nomenclature that encloses the various approaches of this field in [77]. In the following, we revise such a notation while enriching it to make it even more inclusive.

Usually, a cipher is a function f defined over $\mathbb{F}_q$ as:

$$f : \begin{array}{ccc} \mathbb{F}_q^n \times \mathbb{F}_q^m & \rightarrow & \mathbb{F}_q \\ (\underline{x}, \underline{v}) & \mapsto & c \end{array}. \tag{1}$$

where $\underline{x} = (x_1, \dots, x_n) \in \mathbb{F}_q^n$ represents a private key, $\underline{v} = (v_1, \dots, v_m) \in \mathbb{F}_q^m$ represents a public vector and c is the output value. It is possible to have vectors or stream of values as well (e.g., a keystream); in such cases, we consider a function f per output component.

In a natural way, we are able to reformulate the function f as its algebraic normal form (ANF) representation: a polynomial $\mathfrak{p}$ defined in the equivalence classes of the polynomial ring with variables in $x_1, \dots, x_n, v_1, \dots, v_m$ and coefficients in $\mathbb{F}_q$:

$$\mathbb{F}_q[x_1, \dots, x_n, v_1, \dots, v_m] \quad \text{modulo} \quad x_1^q - x_1, \dots, x_n^q - x_n, v_1^q - v_1, \dots, v_m^q - v_m.$$

We omit the modulus for the sake of readability by considering exponents of variables $y_i^{e_i}$ in integers $0 \le e_i \le q-1$. We cast $\mathfrak{p} \in \mathbb{F}_q[\underline{x}, \underline{v}]$ when it is important to distinguish between private and public part, and $\mathfrak{p} \in \mathbb{F}_q[\underline{y}]$, when it is not (with $y = (y_1, \dots, y_N), N = n + m$).

In the following, we use capitalised I and J to identify sets of variable indices in $\{1, \dots, n\}$, $\{1, \dots, m\}$, or $\{1, \dots, N\}$ (as it is clear from the context). Such sets are particularly useful when referring to monomials. This is particularly straightforward in the binary setting $q = 2$ (see later in Sect. 2.3 for the same notation in fields of higher-order $q = p^k > 2$); in fact, monomials $\mathfrak{m} \in \mathbb{F}_2[\underline{y}]$ are in bijective correspondence with subsets $I \subset \{1, \dots, N\}$, $|I| = d$ so that any index set I corresponds to a monomial:

$$\mathfrak{m}_I := \prod_{i \in I} y_i \in \mathbb{F}_2[\underline{y}] \tag{2}$$

where $d := deg(\mathfrak{m}_I) = |I|$.

We then introduce the following notations:

Zero vector ($\underline{0}$) is a generic-length 0 vector, meaning that

$\underline{y} = \underline{0}$ represents $\underline{y} = (0, \dots, 0)$

Unit vector ($\underline{i}$) is a unitary basis vector, meaning that

$\underline{y} = \underline{i}$ represents $\underline{y} = (0, \dots, 0, 1, 0, \dots, 0)$

where the i-th variable only is set to 1.

Unit vector set ($\underline{I}$) is the set of unit vectors obtained from the indices $i \in I$; namely, the underlined notation is mapped through I, i.e.

$\underline{I} := \{\underline{i} | i \in I\}.$

Explicit concat ($::$) is the concatenation notation (omitted when it is not necessary), meaning that:

$(1, 1, 0) :: (1, 0, 1) = (1, 1, 0, 1, 0, 1).$

Vector copy ($\underline{y}^l$) is the notation to l-times repeat y, meaning that

$$\underline{y}^3 = \underline{y} :: \underline{y} :: \underline{y}.$$

Do also note that $\underline{x} = \underline{0}$ is equivalent to $\underline{x} = 0^n$.
Addition $(+)$ is the component wise sum meaning that

$$(1, 1, 0) + (1, 0, 1) = (2, 1, 1).$$

The sum behaviour depends on the underlying finite field; e.g. in $\mathbb{F}_2$, it operates as the `xor`, meaning that

$$1\ 1\ 0 + 1\ 0\ 1 = 0\ 1\ 1$$

where we write binary vectors as bit sequences.
Do also note that the following equality always holds:

$$\underline{x} :: \underline{v} = \underline{x} :: 0^m + 0^n :: \underline{v}$$

Polynomial mapping $\mathfrak{p}(S)$ is a compact notation for applying a polynomial or a function through a set, as it occurs to unit vector sets, meaning that

$$\mathfrak{p}(S) = \{\mathfrak{p}(s) \mid s \in S\}. \tag{3}$$

Partial assignment $(\underline{y}[I \to \underline{a}])$ is a compact notation to perform the assignment of specific variables to the values given in $\underline{a} = \left(a_1, \dots, a_{|I|}\right)$, meaning that

$$\underline{y}[I \to \underline{a}] \qquad \text{represents the assignments} \qquad y_i = a_{j_i} \quad i \in I, \tag{4}$$

where $i \in I$ is mapped to the corresponding element index j_i respecting the natural numbers ordering. The partial assignment is particularly useful when it is combined with (3) to perform *partial polynomial evaluations*[1]:

$$\mathfrak{p}(\underline{y}[I \to \underline{a}]) = \mathfrak{p}(\alpha_1, \dots, \alpha_N) \qquad \text{where} \quad \alpha_i = \begin{cases} a_{j_i} & \text{if } i \in I \\ y_i & \text{otherwise} \end{cases}.$$

The same notation also applies when considering public and private variables separately; consider the following polynomial:

$$\mathfrak{p}(x_1, x_2, x_3, v_1, v_2, v_3) = x_1x_2 + v_1v_2 + x_1v_3 + x_2v_2 \in \mathbb{F}_q[\underline{x}, \underline{v}]$$

then, given two index sets $J = \{1\}$ and $I = \{1, 3\}$, and two vectors $\underline{a} = (1)$ and $\underline{b} = (1, 0)$ we have that $\mathfrak{p}(\underline{x}[J \to \underline{a}] :: \underline{v}[I \to \underline{b}])$ evaluates to:

$$\mathfrak{p}(\underline{x}[\{1\} \to (1)] :: \underline{v}[\{1, 3\} \to (1, 0)]) = x_2 + v_2 + x_2v_2.$$

Cube notation $(\underline{y}[I \to A])$ defines a set of copies of the vector $\underline{y}$ where components specified by I are assigned to every possible value in a given set A, meaning that:

[1] We should say $\alpha_i = a_{j_i}$ if $i \in I$, where j_i is the index of the element of $\underline{a}$ which corresponds to the element i in I. However, we trust in readers' adaptability.

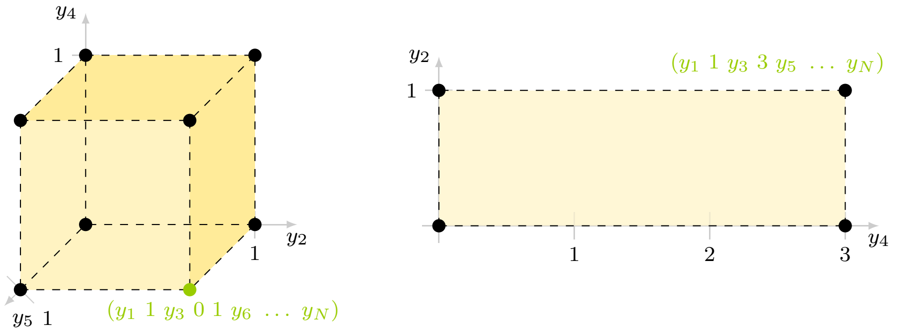
y_4
1
y_2
1
y_5 1
$(y_1\ 1\ y_3\ 0\ 1\ y_6\ \dots\ y_N)$
y_2
1
$(y_1\ 1\ y_3\ 3\ y_5\ \dots\ y_N)$
1
2
3
y_4

Fig. 1 Pictorial representation of the cubes $\underline{y}[\{2,4,5\}]$ (on the left) and $\underline{y}[\{2,4\} \rightarrow (\{0,1\} \times \{0,3\})]$ (on the right) where dots represent points of the cube. One sample point is highlighted and explicitly identified

$$\underline{y}[I \rightarrow A] := \{\underline{y}[I \rightarrow \underline{a}], \ \text{for all}\ \underline{a} \in A\}.$$

The size $d = |I|$ is called *dimension of the cube* and each vector in A has exactly d components. Usually, the set A is built as a cartesian product of binary assignments per each variable:

$$A = A_1 \times A_2 \times \cdots \times A_d \qquad A_i = \{0, a_i\}, \quad a_i \in \mathbb{F}_q.$$

The name *cube* is further legitimised if we consider the case $A = \mathbb{F}_2^d = \{0,1\}^d$. In fact, $\mathbb{F}_2^d$ represents a canonical hyper-cube of dimension d, a very common choice for cube attacks; hence, we omit the A and we write:

$$\underline{y}[I] := \underline{y}[I \rightarrow \mathbb{F}_2^d] = \{\underline{y}[I \rightarrow \underline{a}], \ \text{for all}\ \underline{a} \in \{0,1\}^d\}. \tag{5}$$

To further clarify this cardinal notion, we provide the following two examples whose pictorial representation can be found in Fig. 1:

$$\underline{y}[\{2,4,5\}] = \{y_1 :: z_2 :: y_3 :: z_4 :: z_5 :: y_6 :: \dots :: y_N, \\ \text{for all}\ \underline{z} = \{z_2, z_4, z_5\} \in \{0,1\}^3\}.$$

and

$$\underline{y}[\{2,4\} \rightarrow (\{0,1\} \times \{0,3\})] = \{y_1 :: z_2 :: y_3 :: z_4 :: y_5 :: \dots :: y_N, \\ \text{for all}\ z_2 \in \{0,1\}, z_4 \in \{0,3\}\}.$$

Sum reduce ($\sum S$) is a compact notation for the sum of all the elements within a set S, namely

$$\sum S := \sum_{s \in S} s.$$

The notation fits particularly well with cubes notation:

$$\sum \underline{y}[I] = \sum_{\underline{z} \in \underline{y}[I]} \underline{z} = \sum_{\underline{a} \in \{0,1\}^d} \underline{y}[I \rightarrow \underline{a}].$$

Shorten summation notation can be used to derive the characteristic vector of an index set too, meaning that

$$\underline{y} = \sum \underline{I} \qquad \text{represents} \qquad \underline{y} = (y_1, \dots, y_N) \qquad y_i = \begin{cases} 1 & \text{if } i \in I \\ 0 & \text{otherwise} \end{cases}.$$

Cube & partial assignment ($\underline{y}[J \to \underline{a}, I]$) is a compact notation to perform both partial assignment and cube evaluation $I \cap J = \emptyset$, namely:

$$\underline{y}[J \to \underline{a}, I] \qquad \text{represents} \qquad \underline{y}[J \to \underline{a}, I \to \mathbb{F}_2^d].$$

Do mind that, as we certify at the end of this section, it is common to have the variables with indices in $I^c = \{1, \dots, N\} \setminus I$ set to zero. In such a case, do note that the following equality holds:

$$\underline{y}[I^c \to 0^{N-d}, I] = \underline{0}[I]. \tag{6}$$

Cube & partial assignment in $\mathbb{F}_2$ ($\underline{y}[I_0, I_1, I]$) is an even more compact notation applicable in $\mathbb{F}_2$ to perform both partial assignment and cube evaluation. In $\mathbb{F}_2$, in fact, each variable can only be set to either 0 or 1, therefore it makes sense to distinguish three sets: the set (I_0) of indices of variables substituted by 0, the set (I_1) of indices variables substituted by 1 and the set of indices I of cube variables. We can then shorten the previous notation as follows:

$$\underline{y}[I_0, I_1, I] = \underline{y}[I_0 \to 0^{|I_0|}, I_1 \to 1^{|I_1|}, I].$$

In general, $I_0 \cup I_1 \cup I \neq \{1, \dots, N\}$ therefore the result can still depend of some variables. If it is not the case, however, we remove the set I_0 from the list, since it can be derived from the context, therefore writing:

$$\underline{y}[I_1, I] = \underline{y}[(I_1 \cup I)^c \to \underline{0}, I_1 \to 1^{|I_1|}, I] = \underline{0}[I_1 \to 1^{|I_1|}, I].$$

Finally, if $I_1 = \emptyset$ ($I_0 = \{1, \dots, N\} \setminus I = I^c$), we adopt the strategy of (6):

$$\underline{y}[I^c, \emptyset, I] = \underline{y}[\emptyset, I] = \underline{0}[I]$$

Monomial generation ($\underline{y}^{\underline{s}}$) is a notation to generate a monomial from a characteristic vector $\underline{s} = (s_1, \dots, s_N)$, $s_i \in \{0, 1\}$, in other words:

$$\underline{y}^{\underline{s}} = \prod_{i \in \{1, \dots, N\}} y_i^{s_i}.$$

Note that (2) can be written this way as well:

$$\mathfrak{m}_I = \underline{y}^{\Sigma \underline{I}}.$$

Monomial set generation ($\underline{y}^S$) is the natural mapping of the previous notation to a set S of vectors in $\mathbb{Z}^N$, in other words:

$\underline{y}^S = \{\underline{y}^{\underline{s}} \mid \underline{s} \in S\}$.

Note that $\underline{y}^I$ by monomial set generation is the set of variables y_i such that $i \in I$. This notation is particularly useful in the Division Property setting since, fusing the notation with (5), we get the set of all monomials which divide the monomial $\mathfrak{m}_I$:

$\underline{y}^{\underline{0}[I]} = \{\underline{y}^{\underline{s}} \mid \underline{s} = (s_1, \dots, s_N)$ where $s_i \in \{0, 1\}$ if $i \in I$ and $s_i = 0$ if $i \notin I\}$.

Monomials assignments ($\mathfrak{m}[\circ]$) all the assignment notations given for sets of variables are still valid also for monomials, like the partial assignment $\mathfrak{m}[I \to \underline{a}]$, the cube operator $\mathfrak{m}[I \to A]$ and their combinations.

2.2 Cube attack forefather

Shamir and Dinur introduced in [31] the cube attack by considering any encryption function represented as a polynomial $\mathfrak{p}$ on the binary field $\mathbb{F}_2$. This is a very convenient setting since $z^2 = z$ and $z + z = 0$ for any $z \in \mathbb{F}_2$ and, as in (2), each monomial can be represented by the set of indices of its variables. The attack splits into a key-independent (offline) phase and a key-specific (online) phase.

During the offline phase, the attacker has access to an oracle ciphering machine for $\mathfrak{p}$ and can set $\underline{x}$ and $\underline{v}$ at will; the goal of this phase is to find appropriate values of $\underline{v}$ to get at least n independent linear equations on the $\underline{x}$ unknowns. The online phase takes place when the defender sets a specific key vector $\underline{x}$. Once again, we suppose the attacker as able to set the public vector $\underline{v}$ at will: this assumption requires either a chosen plaintext setting (as it could be for Message Authentication Code generation) or enough spoofing time on randomly generated $\underline{v}$ (as it could be for authentication challenges, e.g., in Wi-Fi handshaking). The goal of this phase is to reconstruct the n equations found during the off-line phase and solve the corresponding linear system to retrieve (a portion of) $\underline{x}$.

We now describe the two phases in detail.

2.2.1 Offline phase

Let $\mathfrak{m}_I$ be the monomial generated by a set of variable indices $I \subseteq \{1, \dots, N\}, |I| = d$. In actual applications, I should address public variables only ($I \subseteq \{1, \dots, m\}$); however, since the methodology we are now describing is general, we prefer (also for ease of notation) to consider I as referring to both private and public variables.

Given the cipher $\mathfrak{p}$ and the monomial $\mathfrak{m}_I$, according the Division Algorithm there exist $\mathfrak{q}_I$ and $\mathfrak{r}_I$ such that:

$$\mathfrak{p}(\underline{y}) = \mathfrak{m}_I \cdot \mathfrak{q}_I(\underline{y}) + \mathfrak{r}_I(\underline{y}) \tag{7}$$

where none of the monomials in the reminder $\mathfrak{r}_I$ is divisible by $\mathfrak{m}_I$ (all of them miss at least a variable from $\underline{y}^I$) and none of the variables $\underline{y}^I$ can be found in the quotient $\mathfrak{q}_I$ (since all of the variables in $\mathfrak{p}$ are of degree 1).

We call $\mathfrak{q}_I$ the *superpoly of I in* $\mathfrak{p}$ and, if $\mathfrak{q}_I$ is linear, we refer to $\mathfrak{m}_I$ as a *d-degree maxterm of* $\mathfrak{p}$.

In particular, applying $\mathfrak{p}$ to the cube $\underline{y}[I]$ exterminates the reminder $\mathfrak{r}_I$ while keeping (when $\underline{y}[I \to 1^d]$) a single instance of the superpoly $\mathfrak{q}_I$. We then claim:

Proposition 1 (cfr. [31]) *The superpoly* $\mathfrak{q}_I$, *defined in* (7), *can be retrieved as*:

$$\mathfrak{q}_I(\underline{y}) = \sum \mathfrak{p}(\underline{y}[I]). \tag{8}$$

Surprisingly, provided that $\mathfrak{m}_I$ is a maxterm, (8) gives us a method to numerically determine the ANF of $\mathfrak{q}_I$, even when $\mathfrak{p}$ is given as a black-box. In fact, since $\mathfrak{q}_I$ is linear it does not contain any of the variables $\underline{y}^I$ and its ANF is given by:

$$\mathfrak{q}_I = a_0 + \sum_{j \notin I} a_j y_j, \qquad a_j \in \{0, 1\}. \tag{9}$$

We then claim:

Proposition 2 (cfr. [31]) *Let* $\mathfrak{q}_I$ *be the superpoly defined in* (7). *Its coefficients, as defined in* (9) *are given by*

$$a_0 = \sum \mathfrak{p}(\underline{0}[I]) \qquad \text{and} \qquad a_j = \sum \mathfrak{p}(\underline{j}[I]) - a_0. \tag{10}$$

As stated above, the linear relations we are building are exploited later in the online phase, where the attacker's aim is to retrieve the private key $\underline{x}$. For this reason, cube variables I are in general chosen amongst the public ones ($I \subset \{1, \dots, m\}$), while the remaining ones ($I^c = \{1, \dots, m\} \setminus I$) are usually tweaked to zero or to any other value $\underline{s}$ to lower the complexity of the resulting system [34]. In this case, the ANF of $\mathfrak{p}$ assumes the following form:

$$\mathfrak{p}(\underline{x} :: \underline{v}[I^c \to \underline{s}]) = \mathfrak{m}_I \cdot \mathfrak{q}_{I,\underline{s}}(\underline{x}) + \mathfrak{r}_{I,\underline{s}}(\underline{x} :: \underline{v}[I^c \to \underline{s}])$$

and, therefore, the superpoly only depends on the private key vector $\underline{x}$:

$$\mathfrak{q}_{I,\underline{s}}(\underline{x}) = \sum \mathfrak{p}(\underline{x} :: \underline{v}[I^c \to \underline{s}, I]).$$

where (10) assume the following form:

$$a_0 = \sum \mathfrak{p}(\underline{0} :: \underline{v}[I^c \to \underline{s}, I]) \qquad \text{and} \qquad a_j = \sum \mathfrak{p}(\underline{j} :: \underline{v}[I^c \to \underline{s}, I]) - a_0. \tag{11}$$

In particular, when $\underline{s} = \underline{0}$, we omit to report the subscript $\underline{s}$, obtaining:

$$\mathfrak{q}_I(\underline{x}) = \sum \mathfrak{p}(\underline{x} :: \underline{0}[I]) \qquad a_0 = \sum \mathfrak{p}(\underline{0} :: \underline{0}[I]) \qquad a_j = \sum \mathfrak{p}(\underline{j} :: \underline{0}[I]) - a_0.$$

2.2.2 Online phase

In online phase we suppose the unknown key $\underline{x} = \underline{k}$ to be set and secret while the public vector $\underline{v}$ to be settable at will. Online phase is made of two parts: (i) for each maxterm $\mathfrak{m}_I$ found in the offline phase, let us obtain an evaluation of the superpoly $\mathfrak{q}_I$ via

$$b_{I,\underline{s}} = \mathfrak{q}_{I,\underline{s}}(\underline{k}) = \sum \mathfrak{p}(\underline{k} :: \underline{v}[I^{\mathrm{c}} \to \underline{s}, I])$$

and (ii) solving the corresponding linear equations system with $\underline{x}$ as unknown variables, yielded by

$$a_0 + \sum a_i \cdot k_i = b_{I,\underline{s}}.$$

The complexity of the first part depends on the size and the number of the cubes, as well as on the practical difficulty to set $\underline{v}$. The second part can be tackled by means of basic linear algebra algorithms in order to retrieve the full key or a portion of it (depending on the number of independent equations found); gaussian elimination algorithm requires e.g., $\mathcal{O}(n^3)$ steps.

Both parts of the online phase, despite being computationally intense, are computationally bounded by the complexity of the offline one. Therefore, being able to carry out the offline phase actually breaks (or weakens) a specific cipher in all of its instances.

2.3 Cube attack in higher order fields

As we highlight in the previous section, the standard cube attack works when polynomials are given over the binary field $\mathbb{F}_2$. This restriction is required to prove (8) which is crucial to the cube attack and derives from fundamental equations in the binary field, namely $y^2 = y$ and $y + y = 0$.

When we place the encryption function in the finite field $\mathbb{F}_q$ those equations assume a different form that depends on the order q and on the characteristic p i.e., if $q = p^k$, then $y^q = y$ and $p \cdot y = 0$. Consequently, monomials are no longer one-to-one with the indices sets since each variable can have exponent up to $q - 1$. A monomial is therefore defined by a vector $\underline{s} = (s_1, \ldots, s_N)$ of exponents where $s_i \in \mathbb{Z}_q$, obtaining the following equation equivalent to (2):

$$\mathfrak{m}_{\underline{s}} = \underline{y}^{\underline{s}} \tag{12}$$

where the set of variables involved are, as always, denoted by I, namely:

$$I = \{i \in \{1, \ldots, N\} \mid s_i \neq 0\}.$$

Given a cipher $\mathfrak{p}$ and a monomial $\mathfrak{m}_{\underline{s}}$, we can derive via Division Algorithm an equation analogous to (7), namely

$$\mathfrak{p}(\underline{y}) = \mathfrak{m}_{\underline{s}} \cdot \mathfrak{q}_{\underline{s}}(\underline{y}) + \mathfrak{r}_{\underline{s}}(\underline{y}) + \mathfrak{r}_I(\underline{y}) \tag{13}$$

where none of the monomials in the reminder is divisible by $\mathfrak{m}_{\underline{s}}$. Though, such monomials can contain some of the variables $\underline{y}^{\underline{I}}$, therefore, in order to resemble an analogy with $\mathbb{F}_2$, it makes sense to divide the reminder into two parts: monomials that do contain all the variables from I ($\mathfrak{r}_{\underline{s}}$) and monomials that do not ($\mathfrak{r}_I$).

Dinur and Shamir claimed in [31] the possibility of extending the attack to a generic field, however, the first proof of this approach can be found in [3] due to Agnesse and Pedicini. Their main contribution consists of reworking Proposition 1 to extend it by considering the relation given by (13):

Proposition 3 (cfr. [3]) *Given a set $\underline{s}$ of exponents working on the variables defined by I, the superpoly defined in* (7) *can be retrieved as*:

$$\mathfrak{q}_I(\underline{y}) = \mathfrak{q}_{\underline{s}}(\underline{y}[I \to 1^d]) + \mathfrak{r}^I_{\underline{s}}(\underline{y}[I \to 1^d]) = \sum \mathfrak{p}(\underline{y}[I]^0) - \sum \mathfrak{p}(\underline{y}[I]^1) \qquad (14)$$

where the set $\underline{y}[I]$ is partitioned as $\underline{y}[I] = \underline{y}[I]^0 \cup \underline{y}[I]^1$ and $\underline{y}[I]^0$ contains those vectors with the same parity as $\underline{y}[I \to 1^d]$.

The parity of $\underline{y}[I \to 1^d]$ is a symbolic parity since it depends on the values assigned to I^c variables, namely:

$$\underline{y}[I]^0 = \{\underline{a} \in \underline{y}[I] \mid \underline{a}[I^c \to \underline{0}] = d \mod 2\}.$$

This key concept is resumed in 2012 Vargiu Master Thesis [103] and later expanded in [76] where Onofri presented many proofs and computational bounds when $\mathfrak{m}_{\underline{s}}$ is chosen as in standard cube attack, i.e. $s_i = 1, i \in I$, see also [13]. If this condition holds, in fact, the cube attack straightforward extends from $\mathbb{F}_2$ to $\mathbb{F}_q$ up to a factor -1 (which depends on the parity of the specific element within the cube evaluation); in fact $\mathfrak{r}^I = 0$, then (14) shortens to

$$\mathfrak{q}_I(\underline{y}) = \mathfrak{q}_{\underline{s}}(\underline{y}) = \sum \mathfrak{p}(\underline{y}[I]^0) - \sum \mathfrak{p}(\underline{y}[I]^1) \qquad (15)$$

and, in particular, we can state, analogously to Proposition 2, that

Proposition 4 (cfr. [76]) *For any polynomial $\mathfrak{p}$ in $\mathbb{F}_q[\underline{x}]$ and cube I yielding a maxterm $\mathfrak{m}_I$, the superpoly has ANF*

$$\mathfrak{q}_I(\underline{y}) = a_0 + \sum_{j \notin I} a_j y_j, \qquad a_j \in \mathbb{F}_q.$$

Coefficients can be numerically evaluated by

$$a_0 = \sum \mathfrak{p}(\underline{0}[I]^0) - \sum \mathfrak{p}(\underline{0}[I]^1) \quad \text{and} \quad a_j = \sum \mathfrak{p}(\underline{j}[I]^0) - \sum \mathfrak{p}(\underline{j}[I]^1) - a_0.$$

An analogous approach to [3] to extend cube attack in higher order fields can be found in [83], where authors fuse standard cube attack with higher order differentiation technique introduced by Lai in [59]. The key concepts are, to the best of our knowledge, totally comparable; however, the processing is performed under the point of view of differentiation techniques. Here, the main contribution is given by the following observation:

Proposition 5 (cfr. [83])

$$\mathfrak{q}_I = \Delta^{(\sum \underline{m})}_{\underline{m} \times \underline{I}} \mathfrak{p} \tag{16}$$

where we are denoting with $\underline{m} \times \underline{I}$ *the multiset of single-variable single-step differentiation*:

$$\underline{m} \times \underline{I} = \{ \underline{i} \text{ taken } m_i \text{ times } \}_{i \in I} \tag{17}$$

and the $\Delta^{(k)}$ *notation is the standard definition of multi-differentiation*:

$$\Delta^{(k)}_{\underline{a}_1, \ldots, \underline{a}_k} \mathfrak{p} = \Delta_{\underline{a}_1} \cdots \Delta_{\underline{a}_k} \mathfrak{p} \tag{18}$$

and the standard differentiation is given by

$$\Delta_{\underline{a}} \mathfrak{p}(\underline{y}) = \mathfrak{p}(\underline{y} + \underline{a}) - \mathfrak{p}(\underline{y}). \tag{19}$$

2.4 Searching for cubes

A cardinal point of the cube attack is to efficiently determine if (9) holds, or, in other words, whether the superpoly $\mathfrak{q}_I$ is linear or not. Two main approaches are used in this context: (i) retrieve the maximum degree δ of $\mathfrak{p}$ and then consider cubes of dimension $d = \delta - 1$ or (ii) employ stochastic tests to guess the linearity of $\mathfrak{q}_I$.

The first approach was firstly introduced and used in [30], where Dinur et al. applied it to attack Keccak sponge functions; however, it has limited applications since correctly determining the degree is hard when the $\mathfrak{p}$ is complex. As we see later in the next section, this approach is often reversed instead, employing the cube attack itself to probabilistically determine δ (see [7]).

The latter approach is the widely used instead. In the original paper [31], Dinur and Shamir employed the Bloom-Luby-Rubinfeld Test from [15], a linearity test originally developed as a self-testing/correcting with applications algorithm. Later, a novel test optimised by reusing computations is proposed in [34]. In this sense, however, a notable contribution is by Winter, Salagean, and Phan who proposed in [111] an improved linearity test based on higher-order differentiation, enhanced by the Moebius transform. Srinivasan et al. propose instead a three-steps algorithm in [87], where filters are applied one after the other to "prove" the linearity of a given black box polynomial at a computational cost of $\mathcal{O}(2^{d+1}(n^2 + n))$.

Testing the linearity of randomly chosen superpolies $\mathfrak{q}_I$ proves however, to be inefficient. For this reason, in [31], variables with indices in I were originally picked accordingly to a random walk on the monomial lattice. "Moving" aleatory, however, does not guarantee a success, therefore efforts were devoted to find a pattern to efficiently select monomials $\mathfrak{m}_I$ while looking for the maxterms. Aumasson et al. proposed in [6] an evolutionary algorithm to search for cubes that maximise the number of rounds after which the superpoly is still unbalanced. Also Wang et al. in [110] propose a new methodology to find more linear equations from the same cube set. Following this trend, Cianfriglia et al. developed in [20] a CUDA framework to parallel control all the cubes within a specific *kite shaped* region of the lattice (see "Appendix A").

However, sophisticated algorithms were also developed to avoid manipulating such large cubes directly. Stankowski in [88], for example, introduced a greedy bit set algorithm with $\mathcal{O}(2^{n+c})$ complexity, later expanded in [53].

By talking about heuristics, cryptographers also tried to reduce the density of the ANFs empirically: two examples can be found in [42, 69].

Following a different research path, more recently, Ye and Tian developed in [117] a novel algebraic criterion to recover the exact superpoly of useful cubes while Rohit et al. introduced in [81] a technique called *partial polynomial multiplication* to obtain complex cubes by multiplying some specific sets of partial polynomials (the degree-homogeneous) from lower round computations.

3 Cube attacks family

The cube attack is a powerful but expensive approach to break ciphers. The idea behind is, however, very solid and flexible and can be combined with many different other approaches to enhance their efficiency. Figure 2 presents various branches traversed by cube attacks.

3.1 Dynamic cube attack and cube testers

The first approach in this sense can be found in [7] where cube building is mixed with efficient property-testers in order to detect non-randomness in cryptographic primitives (or mount cipher distinguishers). The cube framework can, in fact, be exploited in order to test global properties of a black-box polynomial without retrieving the formal expression of the polynomial.

Such strategy is under the name of *Cube Testers* and combines a property tester on the superpoly (for some property $\mathcal{P}$) with a statistical decision rule that probabilistically recognises whenever the superpoly is δ-far from $\mathcal{P}$. Namely, the linearity test exploited in the canonical cube attack is itself a cube tester; other examples of properties realisable as cube testers are polynomial randomness (i.e. the superpoly coefficients are balanced) and the test of presence of neutral variables (i.e. the superpoly does not depend of such a variable).

Cube testers are the basis to create flexible distinguishers as can be seen in [9] first and in [84] later, where authors develop, relying on [88], distinguishers for Trivium based ciphers; however, cube testers main contribution to cryptanalysis can be found in [33] where they are exploited to create *Dynamic Cube Attacks*. The main observation here is that the resistance of many ciphers to cube testers depends on a few number of non-linear operations that usually take place in the latest stages of the encryption process; this is especially true if inputs variables are not mixed enough during the encryption process. Such a behaviour reflects in very few high-order monomials in the ANF of $\mathfrak{p}$ that, if identified at the early stages of the encryption process, can be efficiently killed by vanishing specific input bits—often called *Dynamic variables*. Such dynamic variables, forming a disjoint set from cube

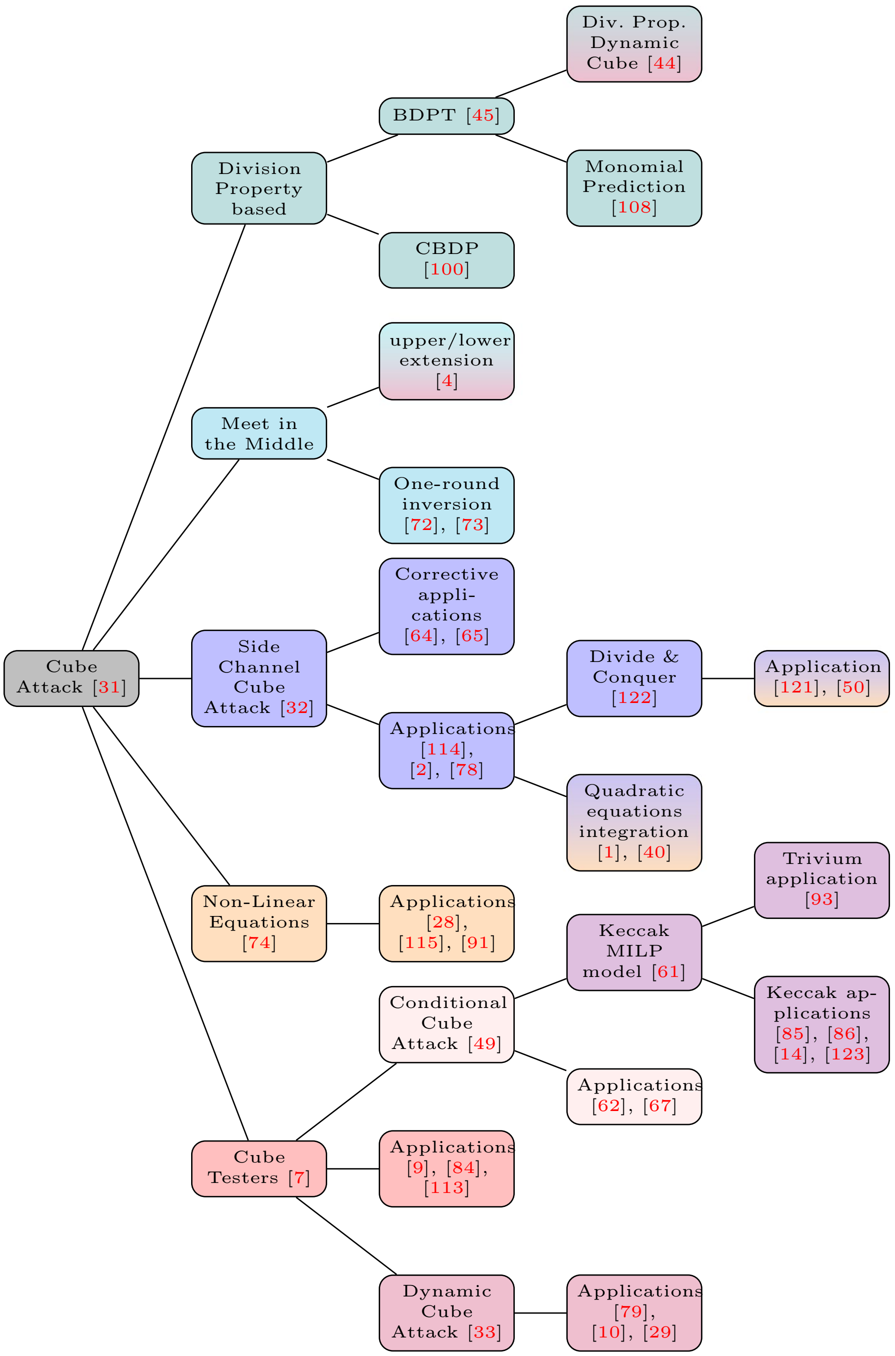
Div. Prop. Dynamic Cube [44]
BDPT [45]
Monomial Prediction [108]
Division Property based
CBDP [100]
upper/lower extension [4]
Meet in the Middle
One-round inversion [72], [73]
Corrective appli-cations [64], [65]
Cube Attack [31]
Side Channel Cube Attack [32]
Divide & Conquer [122]
Application [121], [50]
Applications [114], [2], [78]
Quadratic equations integration [1], [40]
Trivium application [93]
Non-Linear Equations [74]
Applications [28], [115], [91]
Keccak MILP model [61]
Keccak ap-plications [85], [86], [14], [123]
Conditional Cube Attack [49]
Applications [62], [67]
Cube Testers [7]
Applications [9], [84], [113]
Dynamic Cube Attack [33]
Applications [79], [10], [29]

Fig. 2 The cube attack family

variables, usually belong both to public and private ones: the public ones can be set at will during online phases; private ones must instead be guessed and, in particular, these guesses can be confirmed or refuted by cube testers themselves. This, therefore, allows the cryptographer to eventually retrieve key bits without solving any algebraic system at the cost of a more complex offline phase.

Effective usage of this approach can be found, for example, in [79] where authors attack a reduced version of Simon lightweight cipher by using deterministic distinguishers based on cube testers, or in [10] where the author proposes a bi-dimensional dynamic cube attack against 105 rounds Grain v1 that retrieves nine secret-key-bits of the cipher.

A similar approach is also developed in [49] on Keccak sponge functions, where authors combine cube testers with bit-tracing method (see [109]) to create *Conditional Cube Testers*. Here authors impose a further classification of cube variables, dividing those that mix together after the second encryption round (conditional cube variables) from those that are not multiplied with each other after the first round and are not multiplied with any conditional cube variable after the second round (ordinary cube variables).

The same approach is further improved firstly in [62], where the limitation that no mutual multiplication between cube variables occurs in the first round is removed, and then in [67], where more constraints on the number of conditions involving the secret bits are added.

Conditional cube testers were also fused with Mixed Integer Linear Programming by Li et al. in [61] and [14], by Song et al. in [85, 86] and, more recently, by Zhao et al. in [123] where ciphers of the Keccak family were attacked.

Many efforts focussed recently on novel methods for finding cube testers. A possible strategy is to esteem the probability for the superpoly in selected rounds, as in [26]. Another interesting approach can be found in [113] where Liu et al. extended its numeric mapping method for estimating the algebraic degree of NFSR-based cryptosystems (presented in [68]) with the works [53, 88] by Stankowsky et al.

Liu's numeric mapping method was also employed in a more recent work by Kesarwani et al. where, in [55], the authors propose a new algorithm for cube generation following the research branch of [84].

3.2 Exploiting of non-linear equations

As we pointed out in the previous section, linearity is not the only property we can require on the degree of the superpoly. In particular, we can consider maxterms up to a certain small degree and still recover a polynomial system whose resolution is feasible.

As an example, Mroczkowski and Szmidt propose in [74] an improvement to the cube attack concerning both linear and quadratic equations. They employed a Quadracity Test to retain discarded non-linear equations and use key bits obtained via linear equations to solve "by hand" the quadratic ones. This solution highly enhances the number of key bits the attacker can recover while still limiting the cube search phase: in their application to Trivium-709, for example, they claim no brute-force is needed to recover the whole key.

Combining the ideas of [74] with their previous work [110], Wang et al. proposed in [28] a new methodology which makes use of those common variables in two different dimensional cubes to induce maxterms of higher-order from those of lower-order, thus recovering more key bits and reducing the search complexity.

It is also worth of mention the work by Ye and Tian [115], where an experimental approach is employed against Trivium-like ciphers. The authors focus on improving nonlinear superpolys recovery by means of linearisation techniques. Under this setting, several linear and quadratic superpolys are claimed for the 802-round Trivium as well as the possibility of finding a quadratic superpoly for Kreyvium is shown. Relying on specific features discovered on Trivium also an enhanced method to attack Trivium-like ciphers is presented, claiming a generic method of choosing useful nonlinear key expressions.

Clearly, the higher degree equation found this way can be used in many different ways; two more interesting approaches on this side are given by Sun and Guan in [91] where cube attacks are exploited to find new linear relations for linear cryptanalysis purposes and by Eskandari and Ghaemi Bafghi in [39] where non-linear equations are treated as linear equations with noise to attack KATAN lightweight cipher.

3.3 Cube attacks on side channel attacks

The original version of cube attack has no free quarters for uncertainty or measurement errors. However, cube attacks have a natural error correction mechanism (see [32]): by considering a cube K large enough during the offline phase and by evaluating all of its sub-cubes $I \subset K$ yielding linear relations it is in fact possible to gather redundant linear equations. In the online phase, assuming a per-round leakage with uncertainty (as it happens when Hamming weight only is available), the summation of all the leaked bits from a specific sub-cube assignment yields a new linear equation in the $\underline{x}$ with a known term depending on the assignment of the known leaked bits. These new relations can be equated to the corresponding linear combination of key variables $\underline{k}$ pre-evaluated during the offline phase, obtaining a linear system of equations in the $\underline{x}$ and $\underline{k}$ variables that the attacker can exploit.

This approach was applied to many block ciphers by exploiting their specific structure starting from [32], where many linear relations were found for Serpent and AES. Later the same year also Yang et al. used this approach to analyse PRESENT Lightweight cipher in [114].

The same approach led Abdul-Latip et al. to produce two works: in [2] they halved the complexity of NOEKEON block cipher by considering a single bit information leakage from the internal state after the second round; in [1], the authors modified the cube attack used in [114] by employing some low-degree non-linear equation (e.g. quadratic equations) to exploit leakages on PRESENT.

The theory from the previous section was also combined with side channel cube attacks by Fan and Gong in [40] where the security of the Hummingbird-2 cipher (an ultra-lightweight cryptographic algorithm) is discussed. In particular, they describe an efficient term-by-term quadraticity test for extracting simple quadratic equations besides linear ones to be exploited along with the bit-leakage model in a fast GPU model.

Concurrently, also Zhao et al. produced an attack to PRESENT in [122] relying on [114] where a two-layer "divide and conquer" strategy is used concurrently with a sliding window approach and an iterated version of the attack is proposed. Their iterative method was further refined in [121] where the authors also propose a model based on non-linear equations.

During the same year, the full version of LBlock was also attacked by means of these techniques in [50].

Li et al. also approached side channel cube attacks on PRESENT the same year in [64] after their preliminary work on LBlock of the year before [66]. However, their work focussed on data refinement employing the maximum likelihood decoding algorithm in order to correct the side channel outputs by considering it as a linear code transmitted through a binary symmetric channel with crossover probability depending on the accuracy of the measurements. A 50% success rate is achieved in [65] even when data are more than 40% dirty.

3.4 Meet in the middle techniques

One more interesting approach to the cube attack is the possibility to fuse it with *meet-in-the-middle* techniques. Firstly suggested in the original paper [31], the first implementation of this approach is to the iconic 120-bit Courtois Toy cipher (CTC) due to Mroczkowski and Szmidt in [72]. Here the offline phase is performed against four rounds of encryption, by recovering many linear equations as usual (here more than 600 linear equations were found). In the online phase, however, the defender encrypts the messages with a five-round encryption. The explicit inversion is therefore performed by obtaining the ciphertext bits after four rounds of encryption by means of equations in the key bits as unknowns and ciphertext bits as known variables. Due to the simplicity of the encryption round rule, these equations are linear in the key bits so, by equating these polynomials to the one gathered in the offline phase, the result is still a linear system that can be solved as in usual cube attack approaches.

Later on, just mimicking what they did earlier in [72], the authors extended the technique to 255-bit Courtois Toy cipher 2 in [73].

A different approach about dynamic cube attack on stream ciphers that is somehow related to MitM techniques can also be found in [4]. Here Ahmadian et al. proceed in the opposite direction to usual MitM, by explicitly splitting the cipher into three sections computed independently: an upper extension part, an intermediate section where cube variables are chosen, and a lower extension part.

3.5 Cube attacks based on division property

Finally, one of the most recent and promising extensions of cube attack family consists in fusing it with the *division property*, a tool originally introduced by Todo in [97] (later formalised in [98]) as an improvement over Integral Cryptanalysis (see

[57]). A multiset A with elements in $\mathbb{F}_2^N$ is said to have the *division property* $\mathcal{D}_k^N$, with $0 \leq k \leq N$, if:

$$deg(\underline{y}^{\underline{s}}) = \sum \underline{s} < k \quad \text{implies} \quad \sum \underline{y}^{\underline{s}}[\{1, \dots, N\} \to A] = 0$$

at the varying of $s \in \mathbb{Z}_2^N$.

Before pairing it with cube attacks, the concept was firstly extended at FSE 2016, where Todo and Morii in [101] applied it to SIMONs family, introducing the *conventional bit-based division property* (CBDP) and the *bit-based division property using three subsets* (BDPT) and exploiting for the first time the zero-sum property: this solution was more robust than the classical division property, even though it was not efficient enough to carry out a feasible attack.

In order to overcome the efficiency issue, Xiang et al. introduced in [112] the *division trails* i.e. the propagation of the division property through the rounds of the cipher, along with an approach to evaluate them through MILP (Mixed Integer Linear Programming) models, hence enabling faster computations. More formally, given R rounds of a cipher, an input $\underline{y}$ generates, for each round r, an internal state $\underline{y}^{(r)}$. Analogously, a set A with elements in $\mathbb{F}_2^N$, generates R sets $(A^{(0)} = A, A^{(1)}, \dots, A^{(R-1)})$, where $A^{(r)} = \{\underline{y}^{(r)} \mid \underline{y} \in A\}$. A division trail for the set A is a vector $\underline{k} = (k_0, \dots, k_{R-1})$, with $0 \leq k_r \leq N$, such that the division property $\mathcal{D}_{k_r}^N$ holds for the set $A^{(r)}$, for all $0 \leq r < R$. By analysing the round function of the cipher, we can build relations between elements of a trail (i.e., study the propagation of the division property): by writing such relations in a MILP way (relying on the three basic operations of `and`, `xor` and `copy`) we obtain a system of linear inequalities which solutions correspond to valid trails.

The introduction of MILP models allowed Todo *et Al.* in [99] to efficiently apply Division Property along with cube attack, exploiting the CBDP: the non-blackbox representation of the cipher allowed, jointly with the efficient MILP interpretation of the division trail, to obtain unexpected results, hence enabling the authors to break 832-round Trivium. Further results on Trivium were obtained firstly the following year at Crypto'18, where Wang Q. et al. improved the attack up to 839 rounds in [105], and later in 2019 by Wang S. et al. in [107] where, using BDPT, the authors were able to recover the superpolies of 832- and 839-round Trivium with a reduced computational time. It is also worth noticing that, later that year, Ye and Tian showed using MILP-based division property in [116] that many of the key-recovery attacks to Trivium presented at Crypto'18 were also distinguishing attacks.

In [46], Wang et al. introduced a new algorithm to find better cubes: to do so, they used a particular MILP model to find division trails based on SAT from [89] and on the flag technique from [106].

The MILP approach made BDPT exploitable in feasible time too, as it is shown more recently in [45] where up to 841-round of Trivium were successfully broken thanks to a careful selection of which differential trail are found in even number and whose evaluation can be hence safely skipped since their contribution nullifies in the resulting ANF; approaches from [45, 106] well combine together, as it can be seen in [54].

Later, Wang et al. introduced in [108] a novel algebraic version of the division property under the name of *monomial prediction*, also showing its strict similarities with BDPT itself: here, the state variables $\underline{y}^{(r)} = (y_0^{(r)}, y_1^{(r)}, \dots)$ of round r are considered as polynomial components $y_i^{(r)} = \mathfrak{p}_{r,i}(\underline{y}^{(r-1)})$ representing the update function of the i-th component of the state at round r depending on state components at round $r-1$ (hence, $\mathfrak{p}$ can be obtained iterating composition of $\mathfrak{p}_{r,i}$, round-by-round); these formal relations are then exploited to determine whether specific input state variables $y_i^{(0)}$ (or, possibly monomials $\mathfrak{s}$ in the $\underline{y}^{(0)}$ variables) do or do not propagate to the upcoming rounds. This task can be achieved by analysing round-by-round whether $\mathfrak{p}_{r,i}$ does contain first-degree monomials $y_j^{(r-1)}$ for some j or does not, that is, considering the set $P_{r,i}$ of the monomials $\mathfrak{p}_{r,i}$ is made of (namely, such that $\mathfrak{p}_{r,i} = \sum P_{r,i}$), we say that $y_j^{(r-1)}$ is *monomial predicted* at round r if:

$$y_j^{(r-1)} \in P_{r,i} \qquad \text{for some component } j \text{ of the state at round } r-1. \tag{20}$$

A set of variables $(y_{i_0}^{(0)}, y_{i_1}^{(1)}, \dots, y_{i_{R-1}}^{(R-1)})$ such that (20) pairwise holds (i.e., $y_{i_{r-1}}^{(r-1)}$ is monomial predicted in $y_{i_r}^{(r)}$) is said a *monomial trail*. We then claim:

Proposition 6 (cfr. [108]) *A given first-round state variable $y_{i_0}^{(0)}$ can be found in $\mathfrak{p}_{r,i_r}$ if and only if the number of monomial trails connecting them is odd.*

The same proposition holds if considering monomials $\mathfrak{s}^{(0)}$ in $\underline{y}^{(0)}$ variables too.

Given a cube I, Proposition 6 gives us a method to evaluate the superpoly $\mathfrak{q}_I$ of the cube attack by exploiting the monomial trails and, hence, by adopting efficient MILP models. In fact, if we consider the set P of all monomials $\mathfrak{p}$ is made of (namely, $\mathfrak{p} = \sum P$), we can then reformulate (7) as follows:

$$\mathfrak{q}_I = \sum M, \qquad M = (\underline{y}^{0[I \to 1^d, I^c]} \cap P)/\mathfrak{m}_I = \{\mathfrak{m} \in \underline{y}^{0[I^c]} \mid \mathfrak{m} \cdot \mathfrak{m}_I \in P\}.$$

The speed-up obtained via MILP modelling, allowed the authors to break Trivium reduced up to 842-rounds [108].

Further recent work on monomial prediction is by Hu et al. in [47] where the authors describe a novel technique under the name of *nested* monomial prediction to efficiently evaluate the ANF of massive polynomials. In particular, a temporal-complexity pre-evaluation is performed to decide whether a given intermediate state variable should be further expanded or not. Key recovery technique is further enhanced as well by the application of the Möbius transformation.

Division property is also linked with other kinds of cube attacks, such as dynamic cube attacks: in [44], Hao et al. introduced on one hand a heuristic algorithm using flag technique division property that permits to find superpolies with low bias, on the other hand, a new MILP model method for division property using nullification strategies. With this approach, it was possible to define a new dynamic cube attack on Grain-128 with a success probability of 99.83% and to use the new MILP modelling to attack 892 rounds of Kreyvium.

DP was also used to design a heuristic algorithm to find *valuable* cubes, as defined in [92] (i.e., cubes whose superpoly has (at least) a balanced secret variable), that proved to be effective against Trivium and Kreyvium, actually elevating by one round (up to 843 and 893 rounds respectively) the attacks proposed by Hao et al. in [45] and Hu et al. in [48].

4 Frameworks and implementations

Since its very first introduction, cube attack was presented not only as a theoretical attack, but also as a practical methodology to break real-world ciphers.

For this reason, Aumasson et al. built in [6] a first cube tester framework on field-programmable gate array (FPGA) capable of attacking 237 rounds in Grain-128 (out of 256) in 2^{54} cipher runs. The idea behind this implementation is hereafter to split the computation into an input generator, an output collector and a controller unit that employs an evolutionary algorithm for cube searching.

Later FPGA implementation of dynamic cube attacks can also be found in [29] (later revised in [43]) where the RIVYERA computing system is adopted.

The main contribution of previous approaches was however given by the possibility of simultaneously evaluating multiple instances of the cipher in order to fraction execution times. Following this trend, GPUs were for example employed to test SHA-3 candidates against unbalances, as reported in [52].

Cipher evaluations occurring in cube construction are highly related one to the other and often repeated. In [18–20], the Cranic Computing group[2] worked out a complete refactoring of the computation on GPUs in view of repurposing of values already computed. Their main contributions are in the organisation of the cube attack as a Time Memory Data Trade-off algorithm, named *kite attack*, to optimise the computation in accord with the structure of GPU memory layers. The development of a CUDA framework for the cube attack resulted in an open source framework enabling the finding of an 800-rounds superpoly in Trivium [21].

As highlighted by Zhu et al. in [124], the framework development is a key point not only to check attacks feasibility, but also to show the correctness of many unfitting assumptions cryptographers may claim. In particular, their contribution is under a python-based web application (unfortunately no longer accessible by now) to test cube attacks-like (in particular linearity of given superpoly) on different ciphers (Trivium only was implemented, however, simple extensions could be made to integrate other ciphers).

Amongst the most interesting published frameworks, there is also the one for nested monomial prediction by Hu et al. introduced in [47] and discussed in Sect. 3.5.

[2] https://www.cranic.it.

Other notable cube attacks implementations are introduced in [4] as we discuss earlier in Sect. 3.4 and in [50] where Islam et al. develop a GUI toolkit which can load a stream or a block cipher and can check its resistance against the cube attack.

Ye and Tian introduced in [118] a framework for Trivium efficient key-recovery where Stankovski's greedy bit set algorithm fuses with division property and the Improved Moebius Transformation to construct potentially good cubes. Building

Table 1 Results on Trivium cipher

Attack family	Attack type	Rounds out of 1152	Maxterms/ key bits	Time	Bibliography term
Cube attack	Key recovery	735	53 M	2^{30}	[31]
Cube attack	Key recovery	767	35 M	2^{45}	[31]
Cube tester	Distinguisher	790	–	2^{30}	[7]
Cube tester	Non-randomness	885	–	2^{27}	[7]
Cube tester	Distinguisher	806	–	2^{44}	[88]
Cube tester	Non-randomness	1078	–	2^{54}	[88]
Cube tester	Distinguisher	806	–	–	[56]
Non-linear eqs	Key recovery	709	Full	–	[74]
Non-linear eqs	Key recovery	799	Full	2^{39}	[42]
Linear extension	Direct Key Rec	576	26 M	–	[28]
Cube like	Distinguisher	839	–	2^{37}	[69]
Cube like	Key recovery	576	69 M	–	[87]
Cube like	Key recovery	703	–	–	[111]
Bias cube tester	Distinguisher	823	–	$2^{42.74}$	[9, 84]
Cube attack	Key recovery	576	69K	$2^{12.63}$	[51]
Kite attack	Key recovery	799	15 M	$2^{45.3}$	[19]
Kite attack	Key recovery	800	1 M	$2^{46.3}$	[20]
MILP CBDP cube	Key recover	832	–	–	[99]
Non-linear eqs	Key recovery	802	7 M	–	[115]
Division property	Distinguisher	838	–	–	[100]
MILP CBDP cube	Key recovery	839	–	–	[105]
Algebraic recovery	Key recovery	838	5 M	2^{37}	[117]
Cube tester	Distinguisher	850	–	–	[55]
MILP BDPT cube	Key recovery	839	Full	$2^{78.6}$	[48]
MILP BDPT cube	Key revovery	841	–	–	[45]
MILP BDPT cube	Weak key recov	978	1 K	$2^{28.5}$	[119]
MILP BDPT cube	Weak non-random	1108	–	$2^{28.5}$	[119]
MILP monomial pred	Key recovery	842	–	–	[108]
MILP monomial pred	Key recovery	843	2 M	2^{79}	[93]
Div. prop. framework	Key recovery	805	Full	$2^{41.4}$	[118]
Div. prop. framework	Key recovery	806	16	$2^{41.4}$	[118]
Nested monomial pred	Key recovery	845	2 M	2^{78}	[47]
Div. prop. framework	Key recovery	820	30	$2^{53.17}$	[17]

upon their idea, very recently Che and Tian developed in [17] a framework (whose implementation is online available) to find well-balanced superpoly.

Concurrently, Delaune et al. propose in [27] a novel model based on a digraph representation of the cube. Authors claim the graph structure handles some of the monomial cancellations more easily than those based on division property, hence improving timing results. Model implementation is proposed and implemented (online available) in both MILP and Constrained Programming.

Finally, code for the approach adopted by Sun in [92] and by Baudrin et al. in [11] are supplied as well.

5 Applications

The cube attack family focussed since its beginning on stream ciphers like Trivium (Kreyvium, Quavium, ...) and Grain (Grain-v1, Grain-128, ...). We report the respective main results in Tables 1 and 2.

Also PRESENT cipher is entitled of an honourable mention, as many developments in side-channel cube attacks were performed on this cipher. Table 3 reports principal contributions.

Table 2 Results on Grain-128 and Grain-v1 ciphers

Grain-128					
Attack family	Attack type	Rounds out of 256	Maxterms/key bits	Time	Bibliography term
FPGA tester	Distinguisher	237	–	2^{54}	[6]
Cube tester	Distinguisher	246	–	2^{42}	[88]
Cube tester	Non-randomness	Full	–	–	[88]
Dynamic cube	Key recovery	207	80 K	2^{31}	[33]
Dynamic cube	Key recovery	250	Theo	2^{101}	[33]
Dynamic cube	Key recovery	Full	Theo	2^{113}	[33]
Dynamic cube	Key recovery	Full	Full	2^{90}	[29, 43]
Kite attack	Key recovery	160	70,000 M	–	[21]
DP dynamic cube	Key recovery	full	3	$2^{97.86}$	[44]
Cube tester	Distinguisher	191	–	$2^{33.86}$	[26]
Grain-v1					
Attack family	Attack type	Rounds out of 160	Maxterms/key bits	Time	Bibliography term
Cube tester	Distinguisher	90	–	2^{39}	[88]
Cube tester	Non-randomness	96	–	2^{7}	[88]
Cube attack	Key recovery	75	19 M	–	[110]
Dynamic cube	Key recovery	105	9 K	2^{34}	[10]
Dynamic cube	Key recovery	100	Full	2^{47}	[80]

Table 3 Key recovery results via side channel attack on PRESENT cipher with key length of 80 and 128 bit

Attack family	Leakage round	Leaked data	Error toll	Key bits	Time bound	Data required	Biblio. term
PRESENT-80							
Cube attack	3rd	0,1,2,3	0%	48	2^{32}	2^{15}	[114]
Non-linear eqs	After 1	Hamming	0%	64	2^{16}	2^{13}	[1]
Cube attack	3rd	4,8,12	0%	48	–	$2^{11.92}$	[122]
Iterated cube	4rd	0	0%	72	–	$2^{15.154}$	[122]
Non-linear iterated	After 3	Hamming	0%	72	–	$2^{8.95}$	[121]†
Max likeli-hood	After 1	LSB	0.6%	64	$2^{21.6}$	$2^{18.9}$	[64]
Max likeli-hood	After 2	2nd LSB	0.4%	64	$2^{20.6}$	$2^{23.1}$	[64]
Max likeli-hood	After 1	LSB	19.4%	64	$2^{21.6}$	$2^{10.2}$	[64]
Max likeli-hood	After 1	LSB	23.2%	64	$2^{31.6}$	$2^{10.1}$	[65]
Max likeli-hood	After 1	LSB	29.5%	64	$2^{27.6}$	$2^{16.2}$	[65]
Max likeli-hood	After 1	LSB	40.5%	64	$2^{27.6}$	$2^{21.2}$	[65]
PRESENT-128							
Non-linear eqs	After 1	Hamming	0%	64	2^{64}	2^{13}	[1]
Iterated cube	4rd	0	0%	85	–	$2^{15.156}$	[122]
Non-linear iterated	After 3	Hamming	0%	121	–	$2^{9.78}$	[121]†

LSB states the Least significant bit in the hamming weight of the internal state bytes

Error tolerant methods all have success probability above 50%

†: tested on real devices with SC countermeasures like random delay and masking

More recently, much interest was devoted to the lightweight cipher Ascon, one of the finalists of the NIST lightweight cryptography standardisation process that uses a total of 30 rounds of permutations divided into 12/6/12 rounds of initialisation/plaintext processing/finalisation. Different work focused on the different phases of the cipher, actually mounting key-recovery attacks on the initialisation phase, state recovery attacks on plaintext processing phase, forgery attacks on the finalisation phase, or distinguishers on the global permutation function. Many of the recent works assume collateral conditions like the usage of a weak key or the nonce-misuse setting, i.e., a key/nonce pair is reused many times to encrypt (contrary to the recommendations of the designers). Table 4 reports principal contributions, with a notable notion of [36], where the authors design a hardware Trojan to reduce the

Table 4 Results on Ascon-128 lightweight cipher

Attack family	Attack type	Rounds out of (12/6/12)	Data	Time	Bibliography term
Cube-like	Key recovery	$6/\star/\star$	2^{34}	2^{66}	[35]
Cube tester†	Key recovery	$7/\star/\star$	2^{33}	2^{97}	[60]
Cube-like†	State recovery	$\star/5/\star$	2^{18}	2^{66}	[60]
Cube tester†	Forgery	$\star/\star/6$	2^{33}	2^{33}	[60]
Conditional cube	Key recovery	$6/\star/\star$	2^{40}	2^{40}	[63]
Conditional cube	Key recovery	$7/\star/\star$	$2^{77.2}$	$2^{103.9}$	[63]
Conditional cube‡	Key recovery	$7/\star/\star$	$2^{77.2}$	2^{77}	[63]
Cube-like	Key recovery	$7/\star/\star$	2^{64}	2^{123}	[81]
Division property	Distinguisher	7	2^{60}	2^{60}	[81]
Cube-like‡	Key recovery	$7/\star/\star$	2^{64}	2^{97}	[82]
Cube-like‡	Key recovery	$7/\star/\star$	2^{63}	$2^{115.2}$	[82]
Conditional cube†	Part. State Rec	$\star/6/\star$	$2^{44.8}$	$2^{44.8}$	[16]
Conditional cube†	State recovery	$\star/6/\star$	$< 2^{40}$	$< 2^{40}$	[11]

†: Nonce-misuse setting and attack model

‡: Weak-key subspace

Table 5 Results on Keccak sponge function

Attack Family	Attack Type	Rounds out of 24	Time	Memory	Bibliography term
Keccak-MAC-128					
Cube like	Key recovery	6	2^{66}	2^{32}	[30]
Conditional cube	Key recovery	6	2^{40}	–	[49]
Divide-and-conquer	Key recovery	6	2^{45}	2^{13}	[115]
MILP-aided cube-like	Key recovery	6	2^{42}	2^{9}	[14]
Cube-like	Key recovery	7	2^{97}	2^{32}	[30]
Conditional cube	Key recovery	7	2^{72}	–	[49]
Divide-and-conquer	Key recovery	7	2^{84}	2^{64}	[115]
MILP-aided cube-like	Key recovery	7	2^{80}	2^{15}	[14]
Cube-like	Forgery	7	2^{65}	–	[30]
Keccak-MAC-256					
Cube-like	Forgery	8	2^{129}	–	[30]
Keccak-MAC-512					
Conditional cube	Key recovery	6	$2^{58.3}$	–	[61]
Conditional cube	Key recovery	6	2^{40}	–	[86]
Conditional cube	Key recovery	7	2^{111}	–	[85]
Conditional cube	Key recovery	7	$2^{112.6}$	2^{47}	[14]
Conditional cube	Key recovery	7	2^{72}	–	[62]
MILP-aided cube-like	Key recovery	7	2^{108}	2^{108}	[123]

number of initialisation rounds to make cube attack-based key-recovery feasible in 94 s on average.

Even if cube attacks work on ciphers by considering them as black-box polynomials and therefore are suitable to attack nearly any cryptosystem, they can also exploit specific cipher vulnerabilities. It is the case, for example, of the work performed by Dinur and Shamir first, and by many other cryptographers later, on Keccak family (Ketje, Keyak, ...). In Table 5, we report principal results obtained against Keccak sponge function.

Many other cipher were attacked via cube family. It is the case of *e.g.,* lightweight and ultra-lightweight ciphers like SIMONs ([79, 101], ...), Simeck ([120]) KATAN ([56, 108], ...), Subterranean 2.0 ([67]), Hitag2 ([90]), LBlock ([50], ...), Hummingbird-2 ([40]), TinyJAMBU ([37, 38, 94]), MORUS ([46]), Lizard ([54]), and many others.

6 Conclusions

In this paper, we revise and improve a novel notation for the cube attack family. We employ the new notation to analyse and provide a cohesive review of the state-of-the-art for this wide family of cryptanalysis techniques.

We discuss the original Dinur and Shamir's attack in $\mathbb{F}_2$ and we extend it to a generic finite field $\mathbb{F}_q$, also describing recent methodologies employed to find cubes. We summarise the family of attacks in five principal research branches: (i) Cube Testers and their extensions (Dynamic and Conditional Cube Attacks), (ii) Cube Attacks with non-linear equations, (iii) Cube Attacks with information leakages, (iv) Meet in the Middle cube attacks, and (v) Cube Attacks based on the Division Property and its extensions (based on Division Trails and Monomial Prediction). For what concerns the latter, we also, focus on formalising the contributions with the introduced notation, lightening the wordiness of the original one. Later we provide an overview of the few frameworks and implementations currently available. We devote a single appendix to describe in detail our framework implementation of the Kite Attack, where we also present Mickey2.0 as a test case. Finally, we resume by convenient tables all of (to the best of our knowledge) the most significant results obtained through the various approaches applied to the principal attacked ciphers, namely: Trivium, Grain, PRESENT, and Keccak. We believe that cube attacks, in particular, combined with Division Property approaches, still have a long road to run across.

Appendix

Unboxing the kite attack

Here we describe the *Kite-Attack* framework focusing on its source code and how to extend it. The framework has been designed to be cipher independent; as shown in [19] the cost of the attack differs only by a constant factor when different ciphers are used. This "Appendix" aims at providing a detailed guideline on how to extend the framework support to new ciphers; we believe this work can be useful to the crypto-community as the framework provides an easy way to test/analyse ciphers strength w.r.t. the cube attack. A brief description of the structure of the code framework appeared in [22], however here we provide a more detailed version along with all the steps to add a new cipher. We organise the "Appendix" as follows. We start with a brief introduction to Nvidia GPUs and CUDA jargon.[3] Then we describe in detail the framework and its code structure, we define and describe all the steps needed to add a new cipher and, finally, we show how to add it, Mickey2.0, to the framework by crossing these steps.

CUDA and GPU

For a better understanding of our work, we report a few, basic, information about the micro-architecture of NVIDIA GPUs as exposed through the CUDA software framework, since this is the solution used in our study. From a hardware standpoint, an NVIDIA GPU is an array of *Streaming Multiprocessors* (SMs); each SM contains a certain number of CUDA *cores*. From a software perspective, a CUDA program is a sequence of computing *grids*; in turn, each grid is split into *blocks*, and each block comprises a certain number of *threads*. Each function executed on the GPU on behalf of the CPU is called *kernel*. To attain a significant fraction of the theoretical peak performance, *occupancy* (i.e., the fraction of active computing elements at a given time) must be consistently kept high, in such a way that thousands of threads must be ready to be scheduled at any time. Threads are executed by an SM in groups of 32 units called *warps*, and performance improves significantly when threads in the same warp execute the same code with no divergence and access memory according to patterns that privilege *threads* locality, i.e., if threads belonging to the same warp access consecutive memory locations (memory *coalescing* in CUDA jargon). Any thread may access data from multiple memory spaces: *(private) registers*, *(private) local memory*, *shared memory*, *global memory*, and *constant*, *texture* memories that are read-only. *Global memory* is the biggest but slowest memory available and it is persistent across kernel launches by the same application; it can be accessed by all the threads. *Shared memory* is visible to all threads of a block and it has the same lifetime as the block. It is roughly 100× faster than *global memory* and it can be used for caching or to facilitate memory coalescing in cases where it is not possible otherwise [102]. *Local memory* is actually part of the *global memory* and it is used to provide private memory to the threads whenever registers are not enough. *Registers* are the fastest memory and they are also used for the warp-level operation called *shuffle* that allows threads belonging to the same warp to exchange data using registers without passing through higher-latency components of the memory hierarchy.

(α, β)-*Kite attack*

The (α, β)-*kite attack*, introduced in [21], is based on the choice of a set $I_{\max}$ of α public variables and a proper subset I_{min} of β variables, with $\alpha > \beta$. These two sets

[3] Readers familiar with the subject may safely skip Sect. A.1.

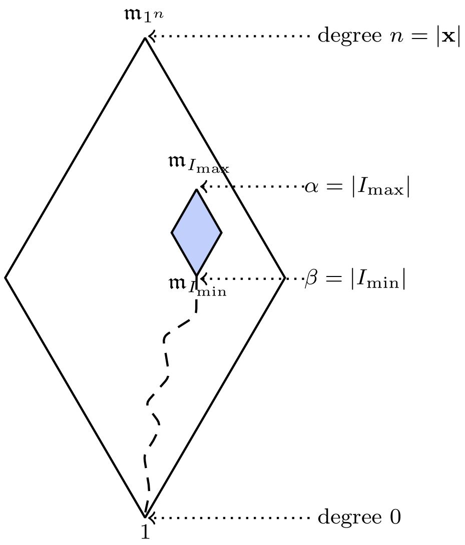
$\mathfrak{m}_{1^n}$
degree $n = |\mathbf{x}|$
$\mathfrak{m}_{I_{\max}}$
$\alpha = |I_{\max}|$
$\beta = |I_{\min}|$
$\mathfrak{m}_{I_{\min}}$
degree 0
1

Fig. 3 A schematic representation of the *kite attack*

represent a *maximal* and a *minimal* cube, respectively $\underline{x}[I_{\max}]$ and $\underline{x}[I_{\min}]$. The name *kite* comes from the observation that the choice of $I_{\min}$ and $I_{\max}$ defines a *diamond-shaped* subspace of all possible monomials, with the bottom vertex of the diamond being $\mathfrak{m}_{I_{\min}}$ and the top vertex being $\mathfrak{m}_{I_{\max}}$. This subspace, schematically depicted in Fig. 3, is made of all monomials $\mathfrak{m}_I$'s such that $I_{\min} \subseteq I \subseteq I_{\max}$ and it is exhaustively explored by our attack.

This definition of the *kite* naturally leads us to a Time Memory Data Trade-Off algorithm where first

(1) for the given minimal index set I_{min} and an initial vector $\underline{v}$, we compute many variants of the cube on the index set $I_{\min}$: one for each possible combination I of the indices in $I_{\max} \backslash I_{\min}$, we evaluate the encryption function in each cube $\underline{x} :: \underline{v}[I, I_{min}]$ for all possible increments of index set $I \subset \{I_{max} \backslash I_{min}\}$ and for any value of $x \in \{0, 1, \dots, n\}$; values are stored in memory to be accessed in a successive moment,

(2) we iteratively combine previously computed results to evaluate coefficients of the superpoly and test its linearity on larger cubes, namely if we want to step from $I = \{i_1, \dots, i_d\}$ to $I' = I \cup \{i_{d+1}\}$, by keeping the setting of remaining variables as specified by the index set I_1 of variables assigned to 1, we apply the following differentiation formula:

$$\sum \mathfrak{p}(\underline{x} :: \underline{v}[I_1^-, I']) = \sum \mathfrak{p}(\underline{x} :: \underline{v}[I_1^+, I]) + \sum \mathfrak{p}(\underline{x} :: \underline{v}[I_1^-, I])$$

where $I_1^- := I_1 \backslash \{i_{d+1}\}$ and $I_1^+ := I_1 \cup \{i_{d+1}\}$ and the increment variable $i_{d+1} \in I_{max} \backslash I$ in such a way that I' always falls in the kite-area ($I_{\min} \subseteq I' \subseteq I_{\max}$).

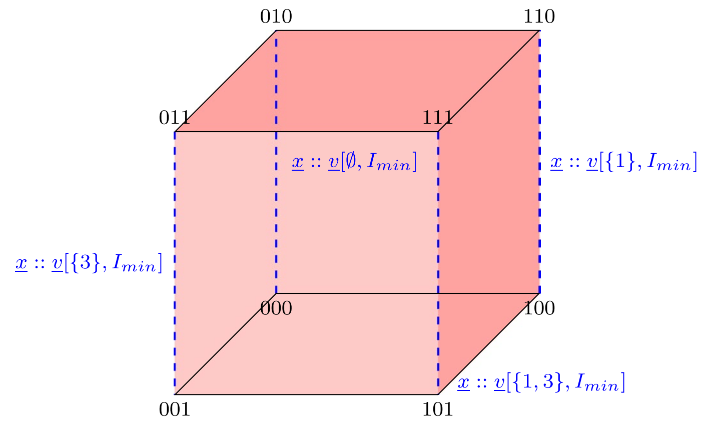
010
110
011
111
$\underline{x} :: \underline{v}[\emptyset, I_{min}]$
$\underline{x} :: \underline{v}[\{1\}, I_{min}]$
$\underline{x} :: \underline{v}[\{3\}, I_{min}]$
000
100
$\underline{x} :: \underline{v}[\{1,3\}, I_{min}]$
001
101

Fig. 4 A schematic representation of the **Kernel1**. For a given index set I_{min} computes all the cubes $\underline{x} :: \underline{v}[I, I_{min}])$ for all possible increments of the index set $I \subset \{I_{\max} \backslash I_{min}\}$. In the picture $|I_{min}| = 1$ therefore any cube on I_{min} contains just two elements and corresponds to the blue dashed edges of the three-dimensional cube

The implementation following this idea leads to two distinct CUDA kernels: **Kernel1** which is responsible for running (1) and **Kernel2** which runs (2).

A schematic representation of the two kernels, in the case of a minimal example with $I_{min} = \{2\}$ and $I_{max} = \{1, 2, 3\}$ is reported in Figs. 4 and 5.

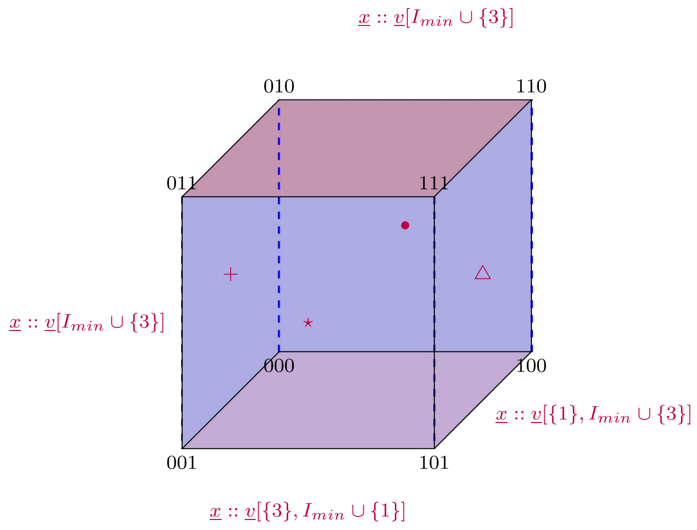
$\underline{x} :: \underline{v}[I_{min} \cup \{3\}]$
010
110
011
111
●
+
△
$\underline{x} :: \underline{v}[I_{min} \cup \{3\}]$
⋆
000
100
$\underline{x} :: \underline{v}[\{1\}, I_{min} \cup \{3\}]$
001
101
$\underline{x} :: \underline{v}[\{3\}, I_{min} \cup \{1\}]$

Fig. 5 A schematic representation of the **Kernel2**. Starting from evaluation results of the first kernel in cubes on I_{min} stored in memory, it combines the results to obtain values on larger cubes. To obtain evaluation of a face we sum results of evaluations on edges: for instance to obtain evaluations on the face labelled as ⋆ which is $\underline{x} :: \underline{v}[\{3\}, I_{min} \cup \{1\}]$: we have to combine results coming from the two edges $\underline{x} :: \underline{v}[\{3,1\}, I_{min}]$ and $\underline{x} :: \underline{v}[\{3\}, I_{min}]$. For each cube it also performs the linearity test by exploiting evaluations of the selected cube with different assignments to $\underline{x}$

Framework code overview

The framework is composed of three source files (`cubaCUDA.cu`, `auxiliary_functions.c`, and `twiddle.c`) along with their corresponding header files. There are two header files: `def.h` which contains all the definitions, macros and includes needed by all the sources, and `key_table.h` must contain two arrays describing how the keys are combined in the linearity tests. Furthermore, each cipher requires the source code for the CUDA implementation and another source file containing auxiliary functions specific to the cipher; for instance, the `setBit` function described below.

The file called `cubeCUDA.cu` includes the two CUDA *kernels*, the main function and other high-level functions useful in managing the attack and several steps of computation concerning superpolys. The file called `auxiliary_functions.c` consists of all the auxiliary functions and wrappers used in the framework. `twiddle.c` contains the functions to generate all combinations of M elements drawn without replacement from a set of N. This code has been written by M. Belmonte and the original version can be downloaded from [12].

A GPU run is the sequential call of the `runAttack` function and then of the `computeSuperpoly` one. The first one is in charge of the real attack, it launches the CUDA *kernels* to compute partial sums over $\underline{x} :: \underline{v}[I_{min}]$ and combines them to test linearity. It also dumps on a binary file all the candidate maxterms found and returns the number of them. The latter function is responsible for reading the binary file containing candidate maxterms, computing the corresponding superpolys, and printing them in human readable format.

Before running the attack, the framework parses the configuration file which contains the following information: the target cipher, the number of initialisation rounds, the indices belonging to $I_{\min}$ and those belonging to $I_{\max} \backslash I_{\min}$, and an ID string to identify the run.

The framework provides some scripts to interactively generate configuration files. After the setup is complete, it verifies the selected CUDA device is able to run the attack and, if so, it generates all the data needed for the attack and copy them on device memory. For instance, it initialises (i) the vector containing the set of keys used in the attack, (ii) the mask representing $I_{\min}$ which is composed by setting the β bits with indexes in $I_{\min}$, and (iii) the $2^{\alpha-\beta}$ masks that represent all the possible monomials which are divisible by $\mathfrak{m}_{I_{\min}}$ and divide $\mathfrak{m}_{I_{\max}}$ (see Fig. 3). Keys and initial vectors (IV) are mapped on contiguous unsigned integer (`u32`); in particular, $\lceil key_{size}/32 \rceil$ and $\lceil IV_{size}/32 \rceil$ unsigned integers are respectively used for each key and IV. Moreover it allocates the memory to store the output of linearity tests or, in the case of the second kernel, to store coefficients of the superpoly.

Every cipher implementation may adopt its own layout to map keys and IV bit indexes. For instance, assuming we have a cipher with IV_{size} of `u32`; the bit iv_0 could be mapped to the most significant bit *(msb)* of the most significant byte *(MSB)*, to the less significant bit *(lsb)* of *MSB* or to the *lsb* of the less significant byte *(LSB)*, and so on. As the framework cannot predict which layout will be used by the cipher, an auxiliary function for each cipher enables the framework to correctly manage any layout. This function basically takes three input parameters: the index of the key/IV to set, the value to set and the pointer to the first unsigned integer that represents the key/IV. We are used to naming these auxiliary functions `setBit`<*cipher name*>. For each supported cipher, the following information has to be provided to the framework through the `def.h` file:

- `KEY_SIZE` and `IV_SIZE`: define the number of bits representing respectively the key and the IV;
- `KEY_ELEM` and `IV_ELEM`: represent the number of `u32` needed to contain respectively one key and one IV;
- `CIPHER_NAME`: is a quoted string containing the cipher name;
- `CIPHER`: is an unquoted string containing the cipher name. This is used to automatically select the `setBit` corresponding to the cipher.
- `KEYS_COEFFICIENT`: is equal to $\texttt{KEY\_SIZE} + 1$. It is used for superpolys computations.
- `TOTAL_KEYS`: represents the smallest multiple of 32 greater than `KEYS_COEFFICIENT`
- `RESIDUAL_KEYS`: contains the value $\texttt{TOTAL\_KEYS} - \texttt{KEYS\_COEFFICIENT}$.

We define some preprocessing macros that automatically select the appropriate `setBit` function once the cipher specific properties are specified in the file `def.h`; of course, the file containing the function implementation should be added to the `Makefile`.

The framework is ready to work with ciphers that support key and IV of length up to 256 bits. We use other preprocessor macros to setup the framework and kernel functions accordingly to the key and IV sizes. We adopt this method as it lets us provide optimised code for any size while, at the same time, it keeps the code simple and easy to read. We use the preprocessor macros also to define which cipher function has to be called by the kernels for the above reason.

Porting the cipher

We now describe the hardest step, adapting the cipher function to CUDA. Given a target cipher E, the first essential step is the definition of the CUDA device function that implements E. This kernel function should require the key and the IV as input parameters and should return the corresponding keystream. If the key cannot be stored in just one `u32` word (i.e. ≥32 bit) it should be provided as multiple `u32` variables[4] rather than an array of `u32` (i.e. it is better `key1`, `key2`, ..., `key` N than `key[` N `]`). In this way, elements of the key are placed in registers (if available) otherwise, data are stored on the global memory which has higher latency access time. Of course, IV should be treated in the same way.

The function implementing the cipher is called by hundreds of threads simultaneously on different inputs; for this reason, this function must be self-contained, i.e. it should use only (thread) local variables to store partial computations and it should not do anything that can interfere with other computations. In other words, our goal is to implement the target cipher E efficiently in CUDA such that it can be executed concurrently by thousands of threads. The implementation should be efficient as the

[4] The number of `u32` words is equal to $\lceil key_{size}/32 \rceil$.

cipher function is called $2^{\alpha} \times 2^{\beta}$ times[5] by each thread involved in the computation; so any effort in optimising it will not be in vain.

To maximise the attack throughput, the cipher function should return 32 bits of the keystream, so to fully exploit the framework's capability to test the linearity of 32 polynomials simultaneously; however, this is usually trivial to do.

Finally, as mentioned in Sect. A.3, the throughput is maximum when all the threads in the same warp execute the same code with no divergence. For this reason, any `if/else` statement should be avoided unless you are sure that the result of the condition is the same at warp level, i.e. all the threads in the warp obtain the same result when testing the condition. In the case this cannot be guaranteed, the `if/else` block of code should be carefully analysed, and, if possible, redesigned with an equivalent block of code that does not contain the branch. An example is reported in Sect. A.6.

Mickey2.0: cipher definition

MICKEY (Mutual Irregular Clocking KEYstream generator) belongs to eSTREAM portfolio. It is an hardware-efficient stream cipher designed by S. Baggage and M. Dodd [8]. It takes two input parameters, an 80-bit secret key K and an IV with variable length between 0 and 80 bits. It is composed of two registers R and S of 100-bits each called respectively the *linear* and *non-linear* registers. It defines two functions `clock_R` and `clock_S` to update R and S respectively. Differently from other ciphers like Trivium and Grain128, Mickey2.0 does not initialise the registers with key and IV; it relies on one specific function instead, called *clock_kg*, that updates both R and S by calling `clock_R` and `clock_S`. In the initial steps, the IV and the key bits are used as input; after these clocks, it runs for 100 more clocks with input 0. Interested readers may find more details in [8].

Mickey2: porting to CUDA

We use as a reference for our porting the *faster* version of Mickey, source code provided in [95]. This version has the advantage that already works with `u32` and efficiently updates register states.

For the other supported ciphers (Trivium and Grain128) we adopted the layout that maps key and IV of index 0 to the *msb* of the *MSB*; to avoid maintaining multiple layouts, we adopt the same layout also for Mickey2.0 and we define the masks representing the update sequences *COMP0*, *COMP1*, *FB0* and *FB1* and the mask defining the *RTAPS* vector accordingly to the selected layout.

With respect to the original implementation, we do not use auxiliary functions for clocking R and S or to initialise the cipher with key and IV; we implement all the steps inside the cipher function to avoid the overhead of calling auxiliary functions. Moreover, for the reasons explained in Sect. A.4, our function does not use arrays for keys and IV but multiple `u32` words; for instance three `u32` for both a key or an IV. All these choices, however, induced us to split the *load IV* and *load key* steps in three loops each. In this way, we duplicate the code but we do not need extra

[5] 2^{α} cubes each of dimension 2^{β}. The cipher function has to be computed for each vertex of the cube.

computations to identify which u32 variable is used at every step. We also define a loop that implements *pre-clock* and another one for the *keystream generator*. This choice allows us to perform optimisations as described below.

The functions CLOCK_KG, CLOCK_R and CLOCK_S as defined in Mickey2.0 specification, contain some if statements. In the following, we analyse each of them:

- CLOCK_KG: the value of the MIXING parameter determines how to compute INPUT_BIT_R. However the result of this check is known *a priori* as it is always TRUE in the initialisation phase and FALSE in keystream generation mode. As we managed the initialisation and keystream generation phases in different loops, we can safely skip the check of MIXING parameter and set the correct value of INPUT_BIT_R in the loops;
- CLOCK_R: there are two if; the first one checks the RTAPS vector to determine which states have to be xored with the value of the FEEDBACK_BIT. The second one checks the value of CONTROL_BIT_R to determine if the new states need to be bitwise-xored with the older ones. We apply the same approach used in [95] in both the checks; we perform a xor operation between states and results of the multiplication of masks representing RTAPS and FEEDBACK_BIT for the first check, and the result of the older states multiplied by CONTROL_BIT_R;
- CLOCK_S: here we have an if-then-else statement. This is a little bit different w.r.t. the other examples mentioned above as we need to manage also the else case. The control statement checks the value of the CONTROL_BIT_S variable; if it is 1, the states of the registers are updated by computing the xor of $\hat{s_i}$ with the result of the multiplication of FB1 and the FEEDBACK_BIT; if it is 0, the xor is computed between the state and the result of the multiplication of FB0 and the FEEDBACK_BIT. We rewrite this check in the following way

  ```
  S0 ^= ( (!contr_s & 0x1) * (S_MASK0_0 * feedback));
   S0 ^= (contr_s * (S_MASK1_0 * feedback));
  ```

  where

  - S0 contains the states $s_0 ... s_{31}$,
  - contr_s is the CONTROL_BIT_S,
  - feedback is the FEEDBACK_BIT, and
  - S_MASK_0_0 S_MASK_0_0 contain respectively $FB0_0$ ...$FB0_{31}$ and $FB1_0$ ...$FB1_{31}$ bits.

  These operations are equivalent to the original if-then-else statement.

The above rewriting of each if statement grants that all the threads of a warp execute the same instruction on different data at the same time. Please notice that if we had left the original if statements we could not have the same assurance. This is due to the fact that values in INPUT_BIT_R, INPUT_BIT_S, CONTROL_BIT_R and

CONTROL_BIT_S are determined from the states of the registers; as each thread executes the cipher on a unique couple of key and IV, each thread may have different values for the variables and consequently yields different results on statement checks.

Framework installation and test case

Here we describe all the steps to install the framework and run a test case. Please notice that you need a Linux computer equipped with an Nvidia GPU of CUDA compute capability ≥ 3.5.

Moreover, you need gcc $\geq$ 4.5.0 and CUDA $\geq$ 7. To verify the CUDA compute capability please refer to [75].

Before starting please download the latest version of the framework from our repository [96]. You can download it as a zip file or you can clone it from the git repository.

A Makefile is provided to install the framework, it instructs the compiler to generate optimised code for most of the compute capabilities. If you have one of the latest GPUs or a Jetson Board, please check if the computing capability of your device is listed on the Makefile; if not, please add it to CUDA_FLAGS variable with the -gencode arch=compute_X, code=sm_X, where X is your computing capability. To install the kite-attack framework, simply run make install.

Once it is successfully installed, you may test it using one of the test configuration files provided in the config directory or you may generate a new configuration file for your customised attack.

An interactive *BASH* script, called genConfigFile.sh, is provided inside the scripts directory. This script helps users to customise their attacks. It allows choosing the target cipher, the number of initialisation rounds to attack, the $I_{\max}$ and $I_{\min}$ indexes, the run identifier and the path-name of the file where the chosen configuration is stored. An example of how to use the script is provided below.

To launch the attack, run the binary file corresponding to the selected target cipher, provide the configuration file, the output directory, and the id of the CUDA device you selected for the attack. If your system has only one device the value to pass is 0, in the case your system has more than one CUDA device provide the id of the chosen device. You may use nvidia-smi tool to obtain the list of all devices of your system along with the id and some other details.

In the following, we provide a complete session as a list of commands, including instructions to get the framework, install it, generate a custom configuration file to attack the Mickey2.0 cipher, and run the attack.

```
$ git clone https://github.com/iac-cranic/kite-attack
$ cd kite-attack
$ make install
```

The interactive script genConfigFile.sh asks the user for some questions and generates the configuration file accordingly to the answers. In the following, we report the list of questions and corresponding answers along with the generated configuration file:

```
$ scripts/genConfigFile.sh
- [Q]: Where do you want to save the configuration?
(default newKiteAttack.conf):
- [A]: testMickey2.conf
- [Q]: Select the target cipher
- [A]: 3
- [Q]: Insert the number of initialization rounds
for the selected cipher: (default 100):
- [A]: 20
- [Q]: Insert run Identifier:
(automatically generated: KITE_xSI6UZjp8m):
- [A]: KITE_MICKEY_TEST_CASE
- [Q]: Insert the value of I_max :
- [A]: 5
- [Q]: Insert the value of I_min :
- [A]: 2
- [Q]: Insert the 0-th value that belongs to I_min
(please note that the indexes start from 0):
- [A]: 0
- [Q]: Insert the 1-th value that belongs to I_min
(please note that the indexes start from 0):
- [A]: 1
- [Q]: Insert the 0-th value that belongs to I_max
but not to I_min (i.e. (I_max\ I_min))
(please note that the indexes start from 0):
- [A]: 2
- [Q]: Insert the 1-th value that belongs to I_max
but not to I_min  (i.e. (I_max\ I_min))
(please note that the indexes start from 0):
- [A]: 3
- [Q]: Insert the 2-th value that belongs to I_max
but not to I_min  (i.e. (I_max\ I_min))
(please note that the indexes start from 0):
- [A]: 4
The configuration file testMickey2.conf has been
successfully generated
=================================
TARGET_CIPHER=Mickey2
INIT_ROUNDS=20
RUN_IDENTIFIER=KITE_MICKEY_TEST_CASE
I_max=5
I_max_minus_I_min=3
I_min=2
I_MAX_SET={0,1,2,3,4}
I_MAX_minus_I_MIN_SET={2,3,4}
I_MIN_SET={0,1}
=================================
```

The last step is to run the attack, this can be done with the following command:

```
$ bin/kite_attack_mickey2 0 testMickey2.conf out_dir
```

In the `config` directory, there are also configuration files to test Trivium and Grain128; with these configuration files the framework finds several superpolys of reduced rounds Trivium and Grain128.

Author Contributions All authors contributed to the study's conception and design. Elia Onofri wrote the first draft of the manuscript and all authors commented on previous versions. Kite-attack development and description ("Appendix A") is due to Marco Cianfrigia. Silvia Onofri has devoted herself to Division Property analysis and description (Sect. 3.5). The work was supervised by Marco Pedicini, who developed the first draft of the new notation and terminology introduced in this paper (Sect. 2.1). All authors read and approved the final manuscript.

Funding Open access funding provided by Università degli Studi Roma Tre within the CRUI-CARE Agreement.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Abdul-Latip, S.F., Reyhanitabar, M., Susilo, W., Seberry, J.: Extended cubes: enhancing the cube attack by extracting low-degree non-linear equations. In: Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, pp. 296–305 (2011). https://doi.org/10.1145/1966913.1966952
2. Abdul-Latip, S.F., Reyhanitabar, M.R., Susilo, W., Seberry, J.: On the security of NOEKEON against side channel cube attacks. Inf. Secur. Pract. Exp. (2010). https://doi.org/10.1007/978-3-642-12827-1_4
3. Agnesse, A., Pedicini, M.: Cube attack in finite fields of higher order. CRPIT **116**, 9–14 (2011)
4. Ahmadian, Z., Rasoolzadeh, S., Salmasizadeh, M., Aref, M.R.: Automated dynamic cube attack on block ciphers: cryptanalysis of SIMON and KATAN. Cryptology ePrint Archive, Paper 2015/040 (2015). https://eprint.iacr.org/2015/040
5. Armknecht, F., Ars, G.: Algebraic attacks on stream ciphers with Gröbner bases. In: Gröbner Bases, Coding, and Cryptography, pp. 329–348. Springer, Berlin (2009). https://doi.org/10.1007/978-3-540-93806-4_18
6. Aumasson, J.P., Dinur, I., Henzen, L., Meier, W., Shamir, A.: Efficient FPGA implementations of high-dimensional cube testers on the stream cipher Grain-128. SHARCS09 (2009). https://eprint.iacr.org/2009/218

7. Aumasson, J.P., Dinur, I., Meier, W., Shamir, A.: Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium. Lecture Notes in Computer Science, pp. 1–22 (2009). https://doi.org/10.1007/978-3-642-03317-9_1
8. Baggage, S., Dodd, M.: The stream cipher MICKEY 2.0, ECRYPT stream cipher submission. www.ecrypt.eu.org/stream/p3ciphers/mickey/mickey_p3.pdf
9. Baksi, A., Maitra, S., Sarkar, S.: New distinguishers for reduced round Trivium and Trivia-SC using cube testers. In: Charpin, P., Sendrier, N., Tillich, J.P. (eds.) WCC2015—9th International Workshop on Coding and Cryptography 2015, Proceedings of the 9th International Workshop on Coding and Cryptography 2015, pp. 1–10. Anne Canteaut, Gaëtan Leurent, Maria Naya-Plasencia (2015). https://eprint.iacr.org/2015/223
10. Banik, S.: A dynamic cube attack on 105 round Grain v1. Appl. Stat. **34**(2), 49–50 (2014)
11. Baudrin, J., Canteaut, A., Perrin, L.: Practical cube attack against nonce-misused Ascon. IACR Trans. Symmetric Cryptol. **2022**(4), 120–144 (2022). https://doi.org/10.46586/tosc.v2022.i4.120-144
12. Belmonte, M.: Twiddle code. Accessed 12 Nov 2020
13. Beyne, T., Canteaut, A., Dinur, I., Eichlseder, M., Leander, G., Leurent, G., Naya-Plasencia, M., Perrin, L., Sasaki, Y., Todo, Y., Wiemer, F.: Out of oddity—new cryptanalytic techniques against symmetric primitives optimized for integrity proof systems. In: Advances in Cryptology—CRYPTO 2020, pp. 299–328. Springer, Berlin (2020). https://doi.org/10.1007/978-3-030-56877-1_11
14. Bi, W., Dong, X., Li, Z., Zong, R., Wang, X.: MILP-aided cube-attack-like cryptanalysis on Keccak Keyed modes. Des. Codes Cryptogr. **87**(6), 1271–1296 (2019). https://doi.org/10.1007/s10623-018-0526-x
15. Blum, M., Luby, M., Rubinfeld, R.: Linearity Testing/Testing Hadamard Codes, pp. 1107–1110. Springer, Berlin (2016). https://doi.org/10.1007/978-0-387-30162-4_202
16. Chang, D., Hong, D., Kang, J.: Conditional cube attacks on Ascon-128 and Ascon-80pq in a nonce-misuse setting (2022). https://eprint.iacr.org/2022/544
17. Che, C., Tian, T.: An experimentally verified attack on 820-round Trivium. In: International Conference on Information Security and Cryptology, pp. 357–369. Springer, Berlin (2023). https://doi.org/10.1007/978-3-031-26553-2_19
18. Cianfriglia, M.: Exploiting GPUs to speed up cryptanalysis and machine learning. Ph.D. Thesis, Roma Tre University (2017/18). http://hdl.handle.net/2307/40404
19. Cianfriglia, M., Guarino, S.: Cryptanalysis on GPUs with the cube attack: design, optimization and performances gains. In: 2017 International Conference on High Performance Computing & Simulation (HPCS), pp. 753–760. IEEE (2017). https://doi.org/10.1109/HPCS.2017.114
20. Cianfriglia, M., Guarino, S., Bernaschi, M., Lombardi, F., Pedicini, M.: A novel GPU-based implementation of the Cube Attack, pp. 184–207. Springer, Berlin (2017). https://doi.org/10.1007/978-3-319-61204-1_10
21. Cianfriglia, M., Guarino, S., Bernaschi, M., Lombardi, F., Pedicini, M.: Kite attack: reshaping the cube attack for a flexible GPU-based maxterm search. J. Crypt. Eng. (2019). https://doi.org/10.1007/s13389-019-00217-3
22. Cianfriglia, M., Pedicini, M.: Unboxing the kite attack. In: La Scala, R., Pedicini, M., Visconti, A. (eds.) De Cifris Cryptanalysis Selected papers from the ITASEC2020 Workshop De Cifris Cryptanalysis: Cryptanalysis a Key Tool in Securing and Breaking Ciphers, *Collectio Ciphrarum*, vol. 1, pp. 31–38. Aracne editrice (2022). https://doi.org/10.53136/97912599486566. https://hdl.handle.net/11590/402925
23. Cid, C., Weinmann, R.P.: Block ciphers: algebraic cryptanalysis and Gröbner bases. In: Gröbner Bases, Coding, and Cryptography, pp. 307–327. Springer, Berlin (2009). https://doi.org/10.1007/978-3-540-93806-4_17
24. Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In: Preneel, B. (ed.) Advances in Cryptology—EUROCRYPT 2000, pp. 392–407. Springer, Berlin (2000). https://doi.org/10.1007/3-540-45539-6_27
25. Courtois, N., Pieprzyk, J.: Cryptoanalysis of block cyphers with overdefined systems of equations. In: Zheng, Y. (ed.) ASIACRYPT 2002, pp. 267–287 (2002). https://doi.org/10.1007/3-540-36178-2_17
26. Dalai, D.K., Pal, S., Sarkar, S.: Some conditional cube testers for Grain-128a of reduced rounds. IEEE Trans. Comput. **71**(6), 1374–1385 (2022). https://doi.org/10.1109/TC.2021.3085144

27. Delaune, S., Derbez, P., Gontier, A., Prud'Homme, C.: A simpler model for recovering superpoly on Trivium. In: Selected Areas in Cryptography: 28th International Conference, Virtual Event, September 29–October 1, 2021, Revised Selected Papers, pp. 266–285. Springer, Berlin (2022). https://doi.org/10.1007/978-3-030-99277-4_13
28. Ding, L., Wang, Y., Li, Z.: Linear extension cube attack on stream ciphers. Malays. J. Math. S. **9**, 139–156 (2015)
29. Dinur, I., Güneysu, T., Paar, C., Shamir, A., Zimmermann, R.: An experimentally verified attack on full Grain-128 using dedicated reconfigurable hardware. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011, pp. 327–343 (2011). https://doi.org/10.1007/978-3-642-25385-0_18
30. Dinur, I., Morawiecki, P., Pieprzyk, J., Srebrny, M., Straus, M.: Practical complexity cube attacks on round-reduced Keccak sponge function. Cryptology ePrint Archive, Paper 2014/259 (2014). https://eprint.iacr.org/2014/259
31. Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. EUROCRYPT **2009**, 278–299 (2009). https://doi.org/10.1007/978-3-642-01001-9_16
32. Dinur, I., Shamir, A.: Side channel cube attacks on block ciphers. Cryptology **2009**, 127 (2009)
33. Dinur, I., Shamir, A.: Breaking Grain-128 with dynamic cube attacks. In: Joux, A. (ed.) Fast Software Encryption, pp. 167–187 (2011). https://doi.org/10.1007/978-3-642-21702-9_10
34. Dinur, I., Shamir, A.: Applying cube attacks to stream ciphers in realistic scenarios. Crypt. Commun. **4**, 217–232 (2012). https://doi.org/10.1007/s12095-012-0068-4
35. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Cryptanalysis of Ascon. In: Topics in Cryptology—CT-RSA 2015: The Cryptographer's Track at the RSA Conference 2015, San Francisco, CA, USA, April 20–24, 2015. Proceedings, pp. 371–387. Springer, Berlin (2015). https://doi.org/10.1007/978-3-319-16715-2_20
36. Duarte-Sanchez, J.E., Halak, B.: A cube attack on a trojan-compromised hardware implementation of Ascon. In: Hardware Supply Chain Security, pp. 69–88. Springer, Berlin (2021). https://doi.org/10.1007/978-3-030-62707-2_2
37. Dunkelman, O., Ghosh, S., Lambooij, E.: Full round zero-sum distinguishers on TinyJAMBU-128 and TinyJAMBU-192 Keyed-permutation in the known-key setting. In: Progress in Cryptology—INDOCRYPT 2022: 23rd International Conference on Cryptology in India, Kolkata, India, December 11–14, 2022, Proceedings, pp. 349–372. Springer, Berlin (2023). https://doi.org/10.1007/978-3-031-22912-1_16
38. Dutta, P., Rajasree, M.S., Sarkar, S.: Weak-keys and key-recovery attack for TinyJAMBU. Sci. Rep. **12**(1), 16313 (2022). https://doi.org/10.1038/s41598-022-19046-2
39. Eskandari, Z., Ghaemi Bafghi, A.: Extension of cube attack with probabilistic equations and its application on cryptanalysis of KATAN cipher. ISC Int. J. Inf. Secur. **12**(1), 1–12 (2020)
40. Fan, X., Gong, G.: On the security of Hummingbird-2 against side channel cube attacks. In: Western European Workshop on Research in Cryptology, pp. 18–29. Springer, Berlin (2011). https://doi.org/10.1007/978-3-642-34159-5_2
41. Faugere, J.C.: A new efficient algorithm for computing Gröbner bases (F4). J. Pure Appl. Algebra **139**(1–3), 61–88 (1999). https://doi.org/10.1016/S0022-4049(99)00005-5
42. Fouque, P.A., Vannet, T.: Improving key recovery to 784 and 799 rounds of Trivium using optimized cube attacks. In: Fast Software Encryption, pp. 502–517. Springer, Berlin (2013). https://doi.org/10.1007/978-3-662-43933-3_26
43. Güneysu, T., Kasper, T., Novotnỳ, M., Paar, C., Wienbrandt, L., Zimmermann, R.: High-performance cryptanalysis on RIVYERA and COPACOBANA computing systems. In: HPC Using FPGAs, pp. 335–366. Springer, Berlin (2013). https://doi.org/10.1007/978-1-4614-1791-0_11
44. Hao, Y., Jiao, L., Li, C., Meier, W., Todo, Y., Wang, Q.: Links between division property and other cube attack variants. In: IACR Transactions on Symmetric Cryptology, pp. 363–395 (2020). https://doi.org/10.13154/tosc.v2020.i1.363-395
45. Hao, Y., Leander, G., Meier, W., Todo, Y., Wang, Q.: Modeling for three-subset division property without unknown subset: improved cube attacks against Trivium and Grain-128aead. In: Lect. N. Computer S., vol. 12105 LNCS, pp. 466–495. Springer, Berlin (2020). https://doi.org/10.1007/978-3-030-45721-1_17
46. He, Y., Wang, G., Li, W., Ren, Y.: Improved cube attacks on some authenticated encryption ciphers and stream ciphers in the internet of things. IEEE Access **8**, 20920–20930 (2020). https://doi.org/10.1109/ACCESS.2020.2967070

47. Hu, K., Sun, S., Todo, Y., Wang, M., Wang, Q.: Massive superpoly recovery with nested monomial predictions. In: Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part I 27, pp. 392–421. Springer, Berlin (2021). https://doi.org/10.1007/978-3-030-92062-3_14
48. Hu, K., Sun, S., Wang, M., Wang, Q.: An algebraic formulation of the division property: revisiting degree evaluations, cube attacks, and key-independent sums (full version) (2020). https://doi.org/10.1007/978-3-030-64837-4_15
49. Huang, S., Wang, X., Xu, G., Wang, M., Zhao, J.: Conditional cube attack on reduced-round Keccak sponge function (2017). https://doi.org/10.1007/978-3-319-56614-6_9
50. Islam, S., Afzal, M., Rashdi, A.: On the security of LBlock against the cube attack and side channel cube attack. In: International Conference on Availability, Reliability, and Security, pp. 105–121. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-40588-4_8
51. Islam, S., Haq, I.U.: Cube attack on Trivium and A5/1 stream ciphers. In: 13th IBCAST, pp. 409–415 (2016). https://doi.org/10.1109/IBCAST.2016.7429911
52. Kaminsky, A.: GPU parallel statistical and cube test analysis of the SHA-3 finalist candidate hash functions. In: 15th SIAM (PP12), pp. 1–15 (2012)
53. Karlsson, L., Hell, M., Stankovski, P.: Improved greedy nonrandomness detectors for stream ciphers. ICISSP (2017)
54. Karthika, S., Singh, K.: Cryptanalysis of stream cipher LIZARD using division property and MILP based cube attack. Discrete Appl. Math. **325**, 63–78 (2023). https://doi.org/10.1016/j.dam.2022.10.011
55. Kesarwani, A., Roy, D., Sarkar, S., Meier, W.: New cube distinguishers on NFSR-based stream ciphers. Designs Codes Cryptogr. **88**(1), 173–199 (2020). https://doi.org/10.1007/s10623-019-00674-1
56. Knellwolf, S., Meier, W., Naya-Plasencia, M.: Conditional differential cryptanalysis of Trivium and KATAN. In: International Workshop on Selected Areas in Cryptography, pp. 200–212. Springer, Berlin (2011). https://doi.org/10.1007/978-3-642-28496-0_12
57. Knudsen, L., Wagner, D.: Integral cryptanalysis. In: Fast Software Encryption, pp. 112–127. Springer, Berlin (2002). https://doi.org/10.1007/3-540-45661-9_9
58. Knudsen, L.R.: Truncated and higher order differentials. In: Fast Software Encryption, pp. 196–211. Springer, Berlin (1995). https://doi.org/10.1007/3-540-60590-8_16
59. Lai, X.: Higher Order Derivatives and Differential Cryptanalysis, pp. 227–233. Springer, Boston (1994). https://doi.org/10.1007/978-1-4615-2694-0_23
60. Li, Y., Zhang, G., Wang, W., Wang, M.: Cryptanalysis of round-reduced ASCON. Sci. China Inf. Sci. **60**(3), 38102 (2017). https://doi.org/10.1007/s11432-016-0283-3
61. Li, Z., Bi, W., Dong, X., Wang, X.: Improved conditional cube attacks on Keccak Keyed modes with MILP method. In: Int. C. Th. Application of Crypt. Information Security, pp. 99–127. Springer, Berlin (2017). https://doi.org/10.1007/978-3-319-70694-8_4
62. Li, Z., Dong, X., Bi, W., Jia, K., Wang, X., Meier, W.: New conditional cube attack on Keccak Keyed modes. In: IACR Transactions on Symmetric Cryptology, pp. 94–124 (2019). https://doi.org/10.13154/tosc.v2019.i2.94-124
63. Li, Z., Dong, X., Wang, X.: Conditional cube attack on round-reduced Ascon. IACR Trans. Symmetric Cryptol. **2017**(1), 175–202 (2017). https://doi.org/10.13154/tosc.v2017.i1.175-202
64. Li, Z., Zhang, B., Fan, J., Verbauwhede, I.: A new model for error-tolerant side-channel cube attacks. In: International Conference on Cryptographic Hardware and Embedded Systems, pp. 453–470. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-40349-1_26
65. Li, Z., Zhang, B., Roy, A., Fan, J.: Error-tolerant side-channel cube attack revisited. In: International Conference on Selected Areas in Cryptography, pp. 261–277. Springer, Berlin (2014). https://doi.org/10.1007/978-3-319-13051-4_16
66. Li, Z., Zhang, B., Yao, Y., Lin, D.: Cube cryptanalysis of LBlock with noisy leakage. In: Kwon, T., Lee, M.K., Kwon, D. (eds.) ICISC 2012, pp. 141–155 (2013). https://doi.org/10.1007/978-3-642-37682-5_11
67. Liu, F., Isobe, T., Meier, W.: Cube-based cryptanalysis of Subterranean-SAE. In: IACR Transactions on Symmetric Cryptology, pp. 192–222 (2019). https://doi.org/10.13154/tosc.v2019.i4.192-222
68. Liu, M.: Degree evaluation of NFSR-based cryptosystems. In: Annual Int. Crypt. C., pp. 227–249. Springer, Berlin (2017). https://doi.org/10.1007/978-3-319-63697-9_8

69. Liu, M., Lin, D., Wang, W.: Searching cubes for testing Boolean functions and its application to Trivium. In: 2015 IEEE ISIT, pp. 496–500. IEEE (2015). https://doi.org/10.1109/ISIT.2015.7282504
70. Mora, T.: The FGLM problem and Möller's algorithm on zero-dimensional ideals. In: Gröbner Bases, Coding, and Cryptography, pp. 27–45. Springer, Berlin (2009). https://doi.org/10.1007/978-3-540-93806-4_3
71. Mora, T.: Solving polynomial equation systems. Cambridge University Press, Cambridge (2015). https://doi.org/10.1017/cbo9781139015998
72. Mroczkowski, P., Szmidt, J.: Cube attack on Courtois toy cipher. Cryptology **2009**, 497 (2009)
73. Mroczkowski, P., Szmidt, J.: The cube attack in the algebraic cryptanalysis of CTC2 (2011)
74. Mroczkowski, P., Szmidt, J.: The cube attack on stream cipher Trivium and quadracity tests. Fund. Inform. **114**(3–4), 309–318 (2012). https://doi.org/10.3233/FI-2012-631. Republish of MroczkowskiSzmidt10
75. Nvidia CUDA GPU capability. https://developer.nvidia.com/cuda-gpus. Accessed 12 Nov 2020
76. Onofri, E.: A computational investigation of the cube attack in general finite fields. Master's Thesis, Roma Tre Univ. (2020). http://bit.ly/3FMXPaN
77. Onofri, E., Pedicini, M.: Novel notation on cube attacks. Collectio Ciphrarum, De Cifris Cryptanalysis, selected papers from the ITASEC2020 workshop (2021). https://doi.org/10.53136/979125994865 65
78. Pang, K.A., Abdul-Latip, S.F.: Key-dependent side-channel cube attack on CRAFT. ETRI J. **43**(2), 344–356 (2021). https://doi.org/10.4218/etrij.2019-0539
79. Rabbaninejad, R., Ahmadian, Z., Salmasizadeh, M., Aref, M.R.: Cube and dynamic cube attacks on SIMON32/64. In: 11th ISC, pp. 98–103 (2014). https://doi.org/10.1109/ISCISC.2014.6994030
80. Rahimi, M., Barmshory, M., Mansouri, M.H., Aref, M.R.: Dynamic cube attack on Grain-v1. IET Inform. Secur. **10**(4), 165–172 (2016). https://doi.org/10.1049/iet-ifs.2014.0239
81. Rohit, R., Hu, K., Sarkar, S., Sun, S.: Misuse-free key-recovery and distinguishing attacks on 7-round Ascon. Cryptology (2021). https://eprint.iacr.org/2021/194
82. Rohit, R., Sarkar, S.: Diving deep into the weak keys of round reduced Ascon. IACR Trans. Symmetric Cryptol. **2021**(4), 74–99 (2021). https://doi.org/10.46586/tosc.v2021.i4.74-99
83. Sălăgean, A., Mandache-Sălăgean, M., Winter, R., Phan, R.: Higher order differentiation over finite fields with applications to generalising the cube attack. Designs Codes Cryptogr (2014). https://doi.org/10.1007/s10623-016-0277-5
84. Sarkar, S., Maitra, S., Baksi, A.: Observing biases in the state: case studies with Trivium and Trivia-SC. Designs Codes Cryptogr. **82**(1–2), 351–375 (2017). https://doi.org/10.1007/s10623-016-0211-x
85. Song, L., Guo, J.: Cube-attack-like cryptanalysis of round-reduced Keccak using MILP. IACR Trans. Symmetric Cryptol. **2018**(3), 182–214 (2018). https://doi.org/10.13154/tosc.v2018.i3.182-214
86. Song, L., Guo, J., Shi, D., Ling, S.: New MILP modeling: improved conditional cube attacks on Keccak-based constructions. In: Int. C. Th. Application of Crypt. Information Security, pp. 65–95. Springer, Berlin (2018). https://doi.org/10.1007/978-3-030-03329-3_3
87. Srinivasan, C., Pillai, U., Lakshmy, K., Sethumadhavan, M.: Cube attack on stream ciphers using a modified linearity test. J. Discrete Math. Sci. Cryptogr. **18**, 301–311 (2015). https://doi.org/10.1080/09720529.2014.995967
88. Stankovski, P.: Greedy distinguishers and nonrandomness detectors. In: International Conference on Cryptology in India, pp. 210–226. Springer, Berlin (2010). https://doi.org/10.1007/978-3-642-17401-8_16
89. Sun, L., Wang, W., Wang, M.: Automatic search of bit-based division property for ARX ciphers and word-based division property. In: ASIACRYPT 2017. Springer, Berlin (2017). https://doi.org/10.1007/978-3-319-70694-8_5
90. Sun, S., Hu, L., Xie, Y., Zeng, X.: Cube cryptanalysis of Hitag2 stream cipher. In: International Conference on Cryptology and Network Security, pp. 15–25. Springer, Berlin (2011). https://doi.org/10.1007/978-3-642-25513-7_3
91. Sun, W.L., Guan, J.: Novel technique in linear cryptanalysis. ETRI J. **37**, 165–174 (2015). https://doi.org/10.4218/etrij.15.0113.1237
92. Sun, Y.: Automatic search of cubes for attacking stream ciphers. In: IACR Transactions on Symmetric Cryptology, pp. 100–123 (2021). https://doi.org/10.46586/tosc.v2021.i4.100-123
93. Sun, Y.: Cube attack against 843-round Trivium. IACR Cryptol. **2021**, 547 (2021)

94. Teng, W.L., Salam, I., Yau, W.C., Pieprzyk, J., Phan, R.C.W.: Cube attacks on round-reduced TinyJAMBU. Sci. Rep. **12**(1), 5317 (2022). https://doi.org/10.1038/s41598-022-09004-3
95. The Mickey2.0 eSTREAM source code. http://www.ecrypt.eu.org/stream/p3ciphers/mickey/mickey_p3source.zip. Accessed 12 Nov 2020
96. The official Kite-attack github repository. https://github.com/iac-cranic/kite-attack. Accessed 12 Nov 2020
97. Todo, Y.: Structural evaluation by generalized integral property. In: Proceedings of EUROCRYPT Part I, pp. 287–314 (2015). https://doi.org/10.1007/978-3-662-46800-5
98. Todo, Y.: Integral cryptanalysis on Full MISTY1. J. Cryptol. **30**(3), 920–959 (2017). https://doi.org/10.1007/s00145-016-9240-x
99. Todo, Y., Isobe, T., Hao, Y., Meier, W.: Cube attacks on non-blackbox polynomials based on division property. In: CRYPTO 2017, pp. 250–279. Springer, Berlin (2017). https://doi.org/10.1007/978-3-319-63697-9_9
100. Todo, Y., Isobe, T., Hao, Y., Meier, W.: Cube attacks on non-blackbox polynomials based on division property. IEEE Trans. Comput. **67**(12), 1720–1736 (2018). https://doi.org/10.1109/TC.2018.2835480
101. Todo, Y., Morii, M.: Bit-based division property and application to SIMON family. In: International Conference on Fast Software Encryption, pp. 357–377. Springer, Berlin (2016). https://doi.org/10.1007/978-3-662-52993-5_18
102. Using shared memory in CUDA C/C++. https://devblogs.nvidia.com/using-shared-memory-cuda-cc/. Accessed 12 Nov 2020
103. Vargiu, M.: Fast algebraic cryptanalysis in finite fields of higher order with the cube attack. In: 100 tesi di crittografia e codici in Italia. 2008-2017, Crittografia book series. Murru, N. and Bartoli, D. and Pavese, F. (2020)
104. Vielhaber, M.: Breaking ONE.FIVIUM by AIDA an algebraic IV differential attack (2007). http://eprint.iacr.org/2007/413
105. Wang, Q., Hao, Y., Todo, Y., Li, C., Isobe, T., Meier, W.: Improved division property based cube attacks exploiting algebraic properties of superpoly. In: CRYPTO 2018 (2018). https://doi.org/10.1007/978-3-319-96884-1_10
106. Wang, Q., Hao, Y., Todo, Y., Li, C., Isobe, T., Meier, W.: Improved division property based cube attacks exploiting algebraic properties of superpoly (FULL VERSION). Lect. N. Computer S. 10991 LNCS (2018). https://eprint.iacr.org/2017/1063
107. Wang, S., Hu, B., Guan, J., Zhang, K., Shi, T.: A practical method to recover exact superpoly in cube attack. Cryptology (2019). https://eprint.iacr.org/2019/259.pdf
108. Wang, S., Hu, B., Guan, J., Zhang, K., Shi, T.: Exploring secret keys in searching integral distinguishers based on division property. IACR Trans. Symmetric Cryptol. **2020**(3), 288–304 (2020). https://doi.org/10.13154/tosc.v2020.i3.288-304
109. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Int. C. Th. applications of Crypt. Tech., pp. 19–35 (2005). https://doi.org/10.1007/11426639_2
110. Wang, Y., Ding, L., Han, W., Wang, X.: The improved cube attack on Grain-v1. Cryptology **2013**, 417 (2013)
111. Winter, R., Salagean, A., Phan Raphael, C.W.: Comparison of cube attacks over different vector spaces. In: Groth, J. (ed.) Cryptography and Coding, pp. 225–238 (2015). https://doi.org/10.1007/978-3-319-27239-9_14
112. Xiang, Z., Zhang, W., Bao, Z., Lin, D.: Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In: ASIACRYPT 2016, 10031 LNCS, pp. 648–678. Springer, Berlin (2016). https://doi.org/10.1007/978-3-662-53887-6_24
113. Yang, J., Liu, M., Lin, D.: Cube cryptanalysis of round-reduced ACORN. In: International Conference on Information Security, pp. 44–64 (2019). https://doi.org/10.1007/978-3-030-30215-3_3
114. Yang, L., Wang, M., Qiao, S.: Side channel cube attack on PRESENT. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) Cryptology and Network Security, pp. 379–391. Springer, Berlin (2009). https://doi.org/10.1007/978-3-642-10433-6_25
115. Ye, C., Tian, T.: A new framework for finding nonlinear superpolies in cube attacks against Trivium-like ciphers. In: Susilo, W., Yang, G. (eds.) Information Security and Privacy, pp. 172–187. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93638-3_11

116. Ye, C.D., Tian, T.: Revisit division property based cube attacks: Key-recovery or distinguishing attacks? IACR Trans. Symmetric Cryptol. **2019**(3), 81–102 (2019). https://doi.org/10.13154/tosc.v2019.i3.81-102
117. Ye, C.D., Tian, T.: Algebraic method to recover superpolies in cube attacks. IET Inf. Secur. **14**(4), 430–441 (2020). https://doi.org/10.1049/iet-ifs.2019.0323
118. Ye, C.D., Tian, T.: A practical key-recovery attack on 805-round Trivium. In: Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part I, pp. 187–213. Springer, Berlin (2021). https://doi.org/10.1007/978-3-030-92062-3_7
119. Ye, C.D., Tian, T., Zeng, F.Y.: The MILP-aided conditional differential attack and its application to Trivium. Des. Codes Cryptogr. p. 89 (2020). https://doi.org/10.1007/s10623-020-00822-y
120. Zaheri, M., Sadeghiyan, B.: SMT-based cube attack on round-reduced SIMECK32/64. IET Inf. Secur. **14**(5), 604–611 (2020). https://doi.org/10.1049/iet-ifs.2019.0496
121. Zhao, X., Guo, S., Zhang, F., Wang, T., Shi, Z., Liu, H., Ji, K., Huang, J.: Efficient hamming weight-based side-channel cube attacks on PRESENT. J. Syst. Softw. **86**(3), 728–743 (2013). https://doi.org/10.1016/j.jss.2012.11.007
122. Zhao, X.J., Wang, T., Guo, S.: Improved side channel cube attacks on PRESENT. Cryptology **2011**, 165 (2011)
123. Zhao, Z., Chen, S., Wang, M., Wang, W.: Improved cube-attack-like cryptanalysis of reduced-round Ketje-Jr and Keccak-MAC. Inf. Process. Lett. **171**, 106124 (2021). https://doi.org/10.1016/j.ipl.2021.106124
124. Zhu, B., Yu, W., Wang, T.: A practical platform for cube-attack-like cryptanalyses. Cryptology **2010**, 644 (2010)

Authors and Affiliations

Marco Cianfriglia[1] · **Elia Onofri**[1] · **Silvia Onofri**[2] · **Marco Pedicini**[1]

Marco Cianfriglia
mcianfriglia@uniroma3.it

Silvia Onofri
silvia.onofri@sns.it

Marco Pedicini
mpedicini@uniroma3.it

[1] Department of Mathematics and Physics, Roma Tre University, Largo San Leonardo Murialdo 1, 00146 Rome, Italy

[2] Scuola Normale Superiore, Piazza dei Cavalieri 7, Pisa, Italy