Algebraic Properties of the Cube Attack

Frank-M. Quedenfeld[1], Christopher Wolf[2]

[1] University of Kassel, Germany
frank.quedenfeld@mathematik.uni-kassel.de,frank.quedenfeld@googlemail.com
[2] Ruhr University Bochum, Germany
Christopher.Wolf@ruhr-uni-bochum.de,chris@Christopher-Wolf.de

Abstract. Cube attacks can be used to analyse and break cryptographic primitives that have an easy algebraic description. One example for such a primitive is the stream cipher *Trivium*. In this article we give a new framework for cubes that are useful in the cryptanalytic context. In addition, we show how algebraic modelling of a cipher can greatly be improved when taking both cubes and linear equivalences between variables into account. When taking many instances of Trivium, we empirically show a saturation effect, *i.e.* the number of variables to model an attack will become constant for a given number of rounds. Moreover, we show how to systematically find cubes both for general primitives and also specifically for Trivium. For the latter, we have found all cubes up to round 446 and draw some conclusions on their evolution between rounds. All techniques in this article are general and can be applied to any cipher.

Keywords: Trivium, cubes, algebraic modelling, cube testing, similar variables, cube classification

Version: 2013-11-29

1 Introduction

Cube attacks were proposed back in 2008 by *Dinur and Shamir* in [DS09a]. They can be seen as a generalization of *Lai's* Higher Order Differentials [Lai94] and *Vielhaber's* AIDA [Vie07]. Initially, they were applied against Trivium, one of the stream ciphers in Ecrypt/eSTREAM portfolio 2 (hardware oriented ciphers) [CP08]. Although many cryptanalytic attempts, Trivium is still unbroken. But even back in their initial paper, the article [DS09a] did put cubes into a much wider framework to attack any symmetric cryptographic primitive. Consequently, both block ciphers and hash functions were attacked with cubes in [ADMS09, DS09b, YLWQ09, MS09, Lat09].

Still, cube attacks enjoy most success when applied against stream ciphers. In particular, a version of the stream cipher Grain-128 [HJMM06], was broken using cube attacks [DAS11]. It should be noted that Grain-v1 is also part of portfolio 2 of the eSTREAM project. In addition, a cube attack is able to recover the full key of a 799−round-reduced version of Trivium in about 2^{62} operations [FV13]. When working with cube attacks, Trivium remains the standard test bed because of its simple and efficient yet secure structure.

In literature, there are different and conflicting definitions of the term "cube". We mainly follow the definitions from *Dinur and Shamir* [DS09a] and only refer the reader to alternative definitions from [Vie07]. There are further generalizations including testing for cubes with degree two [MS11] or to test for other properties that are easier to find and help us to distinguish the cipher from random bit streams [ADMS09]. Cubes were also used to directly lower the degree of the update function in a cipher [DAS11]. This has greatly improved the efficiency of this attack.

1.1 Organization and achievement

In section 2 we classify different types of cubes. In particular, we give a cleaner and more complete definition of the different variants of cubes than available in contemporary literature. We hope that this stipulate further research in concrete attacks against concrete stream ciphers. In addition we introduce new variants of cubes, namely state cubes, factor cubes, and polycubes. State cubes are cubes in the initialization phase of a cipher while factor cubes replace the cube *monomial* by a cube *polynomial*. This can be viewed as a generalization of Vielhaber's cancellation attack [Vie08]. In all cases, we give concrete examples to show their existence. All in all, these new variants become particularly interesting when combined with a new algebraic modelling technique from section 4.

Algebraic modelling of non-linear ciphers as given in the literature does not achieve its full potential yet. In [SFP08, T+13, SR12] the algebraic representation of Trivium stays the same as in [Rad06]. This means that all state bits of one Trivium instance equal exactly one variable, even after the initilization phase. The model described there can neither handle more than one Trivium instance nor does it take advantage of cubes within the initialization phase. We introduce new modelling techniques using linear algebra and thus build a more general model of Trivium to overcome these disadvantages (cf. section 4). Main advantage of our alternative algebraic representation is the use of several instances of a given cipher. Thus reducing the number of variables and quadratic monomials, using linear algebra techniques. In addition, we observe a *saturation* property where our model does not need more intermediate variables although we increase the number of instances and hence output bits. The modelling technique presented here is quite general and not limited to the Trivium cipher. This new algebraic modelling makes direct use of state cubes. We hence introduce new ways of cube finding in section 5. In a nutshell, we have a deterministic algorithm that can be used to find *all* cubes up to a certain round, as opposed to the existing probabilistic algorithms. Using our implementation of the deterministic algorithm, we can cover all cubes up to round 446 for output cubes and 397 for state cubes. By restricting our search space in the IV or key variables, we develop two algorithms that can go up to arbitrary rounds—at the cost of not finding all available cubes anymore.

The results we achieve with these algorithms are discussed in section 6. In particular, we give the number of cubes in the output and in the state, the size of the output function in Algebraic Normal Form or ANF (counted in the overall number of monomials), and also the distribution of cubes by size. This leads us to formulate a conjecture about the life-cycle of cubes in Trivium.

This article concludes with section 7. In the appendix, we list the number of cubes per round and dimension in the output up to round 446.

2 An overview of cubes

We discuss now several possible and useful definitions of *cubes*. Unfortunately, parts of the literature are not too precise on this topic. Evidently, this makes it difficult to compare results between papers. We start this section by the most easy definition of cubes. A similar definition has already been used in the original article [DS09a].

2.1 First glance at cubes

Let $\mathbb{B} := \{0, 1\}$ the field of size 2 and $n := n_v + n_k$ the total number of variables (n), the number of IV variables (n_v), and the number of key variables (n_k). Moreover, we have two distinct sets of

variables $V := \{v_1, \ldots, v_{n_v}\}$ (IV variables) and $K := \{k_1, \ldots, k_{n_k}\}$ (key variables) Based on this we define the function

$$f : \mathbb{B}^n \to \mathbb{B}\,.$$

We also write $f(V, K)$ (or $f(K, V)$) to stress on which set of variables we work. We know that there is a unique algebraic normal form (ANF) over $K \cup V$ for this function f. Let $M \subset \{\mu \subset (V \cup K)\}$ be a set of monomials. By convention, we set $x_\mu := \prod_{a \in \mu} a$ and also $\prod_\emptyset := x_\emptyset := 1$. Then we can write the ANF of f as

$$f(V, K) = \sum_{\mu \in M} \prod_{x \in \mu} x\,.$$

Viewing monomials as sets of variables and and functions as sets of monomials, we also write $\mu \in f$ instead of $\mu \in M$ and $u \in x_\mu$ for some variable $u \in \mu$.
Let $C \subset V$ be a subset of the IV variables and $x_C := \prod_{a \in C} a$ a monomial. Then we can write

$$f(V, K) = x_C p(K) + r(V, K)$$

for $p(K)$ a polynomial solely in the key variables K and a residual polynomial r over all variables $V \cup K$. If $\nexists \mu \in r : x_C | \mu$, we call C a *cube* and x_C the corresponding *cube monomial*. If unambitious, we will also say that x_C is a *cube* for short. In addition, $p(K)$ is called the *superpoly* of the cube C. By its very definition, it gives information on the key variables of the function f. We call the degree $\deg(p)$ of the superpoly the *key-degree of a cube* and the number of elements $c := |C|$ in C the *dimension of the cube*. Note that this coincides with the degree of the cube monomial $\deg(x_C)$ so we have c=$\deg(x_C)$. For the k-dimensional cube C let I_C be a subset of $\mathcal{P}(V)$ including all 2^k vectors from $\mathbb{B}^{|C|}$. In essence, we assign all the possible combinations of 0/1 values to the variables in C.
In [DS09a] the following observation of the above construction has been proven.

Theorem 1. *For any polynomial p and cube C it holds*

$$p(K) = \sum_{v \in I_C} f(v, K).$$

Proof. Consider the decomposition $f(V, K) = x_C p(K) + r(V, K)$. We first examine an arbitrary term $\mu \in r(V, K)$. Since C is a cube we have $x_C \nmid \mu$. So there exists at least one IV variable in C that is missing in μ. As a consequence in $\sum_{v \in I_C} f(v, K)$ it is added an even number of times. As we work in GF(2), these terms cancel out.

Next we examine the polynomial $x_C p(K)$. All vectors $v \in I_C$ make the monomial x_C zero except the all-one-vector $(1, \ldots, 1)$. This implies that the polynomial $p(K)$ appears in the above sum only once, namely when all variables from x_C are set to 1. □

According to theorem 1, once we have found a superpoly we can add the output of the function f to obtain its concrete value from the set $\mathbb{B}$. As we know that this value is equal to the value of the superpoly p, we obtain exactly one equation of key-degree $\deg(p)$. Having and solving a full system of these equations leads to information about the key of the cipher. So the cube attack consists of two phases: by first finding cubes and consequently superpolys in a given cipher *(offline phase)* we are then able to relate these equations to some given output *(online phase)* and reconstruct the secret key values by solving the corresponding system of equations. *Note:* Even if we only reconstruct a part of the key bits, say 1/2 or even 1/4 of them, we have nevertheless found a valid cryptographic attack against this cipher when brute-forcing the rest of the key-space.

Example 1. Let $K := \{a, b, c\}, V := \{\alpha, \beta, \gamma\}$ be the sets of key and IV variables, respectively. Moreover, consider the following function $f : K \cup V \to \mathbb{B}$ in algebraic normal form (ANF):

$$f(K, V) := ab\alpha + a\beta\gamma + b\beta\gamma + bc\alpha + bc\beta + abc\alpha\gamma + bc + a + b + \beta\gamma + 1$$

Using the above definition, we have the following superpolys, indexed by IV sets:

IV set	superPolys	linear
$\emptyset$	$bc + a + b + 1$	-
α	$ab + bc$	-
β	bc	-
$\alpha\gamma$	abc	-
$\beta\gamma$	$a + b + 1$	$\star$

Only the superpoly for the cube $\beta\gamma$ is linear and therefore directly usable for a crytanalytic attack.

2.2 Informal treatment

In this chapter we take a look of some special cases when dealing with the cube attack. First, we look at the degree of the superpoly:

constant cubes: the degree of the superpoly is -1 ***(negative cube)*** or 0 ***(one cube)***.
linear cubes: the degree of the superpoly is exactly 1.
higher order cubes: the degree of the superpoly is higher than 1.
quadratic cubes: the degree of the superpoly is between 1 and 2.
cubic cubes: the degree of the superpoly is between 1 and 3.

In addition, we can look at the number of free IV variables:

fixed cubes / zero cubes: When we fix all IV variables that are not in a cube, we call it a *fixed cube*. If all these variables are fixed to zero, it is a *zero cube*.
free cubes: In contrast, if all IV variables outside of the cube can take on arbitrary values and the cube property remains, we call it a *free cube*.
flexible cubes: All cases in between, *i.e.* we need to fix some IV variables to arbitrary values to ensure the cube property, but not all.

Furthermore, instead of looking for cubes in the output of a cipher we can search for cubes in its update function. These so-called **state cubes** can be used to simplify the internal state of the cipher.

From cryptanalysis, we know that setting specific variables or functions to certain values can drastically reduce the amount of computations we need for a cube attack. This is captioned in the definition of **dynamic cubes**.

An alternative way to reduce the workload are **mixed cubes**. Here, we include parts of the IV variables into the superpoly.

Another way to reduce the degree of the superpoly is summing up several cubes such that their high degree monomials cancel out. This is called a **polycube**.

Another relaxation is the **factor cube**. Here, we replace the cube monomial by a *polynomial*.

Cube Testers. For the sake of completness we also mention "cube testers" from [ADMS09]. Instead of looking at *algebraic*, its authors look at *statistical* properties of cubes. For example, they are concerned if the output of the superpoly $p(K)$ is actually balanced. As the treatment of cubes in this article is more algebraically oriented, we not go further in this topic but refer the interested reader to the original article instead.

2.3 Cubes by degree

Let d be the key-degree of a cube. Then we distinguish the following cases: $d = 1$: *linear cube*, $1 \leq d \leq 2$: *quadratic cube*, $1 \leq d \leq 3$: *cubic cube*, and $d \geq 2$: *higher order cube*. In addition, we have *constant cubes* for $d \leq 0$. If we need to distinguish between $d = 0$ and $d = -1$, we call the corresponding cubes a *one cube* or *negative cube*, respectively.

Example 2. As in example 1, let $K := \{a, b, c\}, V := \{\alpha, \beta, \gamma\}$ be the sets of key and IV variables, respectively. In addition, we consider $f : K \cup V \to \mathbb{B}$ as

$$f(K, V) := ab\alpha + a\alpha\beta + a\beta + a\beta\gamma + b\beta\gamma + bc\alpha + bc\beta + abc\alpha\beta + abc\alpha\gamma + bc + a + 1 + \beta\gamma + \gamma$$

This leads to the following cubes:

cube	superpoly	degree	classification
$\emptyset$	$bc + a + b + 1$	2	quadratic
α	$ab + bc$	2	quadratic
β	$bc + a$	1	**linear**
γ	1	0	***constant*** or ***one***
$\alpha\beta$	$abc + a$	3	*cubic*
$\alpha\gamma$	abc	3	*cubic*
$\beta\gamma$	$a + b + 1$	1	**linear**
$\alpha\beta\gamma$	0	-1	***constant*** or ***negativ***

As we can see, using the superpolys from the linear cubes alone we cannot determine the values of all key variables a, b, c. To this aim, we need to combine the linear cubes $\{\beta, \beta\gamma\}$ with at least one higher order cube from the set $\{\alpha, \alpha\beta, \alpha\gamma\}$. Note that $\emptyset$ does not correspond to a cube in the strict sense. On the other hand, this special case does not do any harm either, so it is included in our definitions.

While constant cubes do not help to recover specific key bits, they are invariants within the cipher and can help in algebraically modelling the cipher. By creating more equations, they aid solving the corresponding system. The name "negative cube" is motivated as it corresponds to a "monomial not present", cf. example 2.

In addition, it is debatable if higher order cubes are of real interest for cryptanalysis as solving quadratic equations is considered to be a hard problem. However, at least for quadratic cubes and cubic cubes, this should be the case for Trivium as the following numbers show: $\binom{80}{2} + 80 = 3240$ and $\sum_{i=1}^{3} \binom{80}{i} = 85,400$. So we need to collect around 3200 linearly independent superpolys of degree at most 2 and around 85000 superpolys of degree at most 3 to recover all key variables of Trivium by simple linearization. This does not take into account faster algorithms as F_5 that work quite well for the number of equations close to the above numbers. Hence using them will allow us to reduce the number of cubes needed even further. In addition, we might combine of linear equations with quadratic equations. By eliminating variables, we obtain a quadratic system in less unknowns that is usually easier to solve. Finding actual cubes will be dealt with in Section 5.

2.4 Cubes by fixing

The above definition of a cube is a bit misleading as it makes us belief that we are working with a full polynomial over all variables from $K \cup V$ here. In reality, this is usually not the case as a full algebraic description of the function f is far too large to be represented explicitly. In the case of Trivium, we have plotted the number of monomials in figure 4, cf. Section 6.1 for more details. Hence it is custom to fix *all* IV variables from $V \backslash C$ to arbitrary values, in particular to zero. We capture this (and the other extreme case—no fixing at all) in the following definitions.

Let $S \subset V$ be the *set of set (fixed) variables*, $F \subset V$ the *set of free variables* such that $S \cup F = V \backslash C, S \cap F = \emptyset$ and consequently $|S| + |F| = |V \backslash C|$. Note that this also includes the two cases $S = \emptyset$ or $F = \emptyset$. We call $f := |F|$ the *degree of freedom of a cube*. Moreover we have $\alpha : S \to \mathbb{B}$ an *assignment* of the set variables. If we have a strict ordering of the IV variables, we can replace the assignment α with a vector $a \in \mathbb{B}^{|S|}$. We now restrict the function f according to this assignment α and obtain

$$f(\alpha, V, K) = f(a, V, K) = f(F, K) = x_C p(K) + r(F, K)\,.$$

Note that the definition of a cube is relaxed this way. In particular, assigning the zero-value to a variable v_i removes all monomials $\mu \in f$ with $v_i | \mu$ from the restricted function $f(F, K)$. This will become a central argument in the proofs of Section 5. Based on the above definition, we distinguish the following cases:

fixed cubes: the degree of freedom is zero, *i.e.* we have $F = \emptyset$, $S = V \backslash C$, and $f = |F| = 0$.
zero cubes: as *fixed cubes*, but additionally the assignment is the all-zero function $\alpha : I \to 0$.
free cubes: here no variable is fixed at all, *i.e.* we have $F = V \backslash C$, $S = \emptyset$, and $f = |V \backslash S|$.
flexible cubes: we have $1 \leq f < |V \backslash S|$ for the degree of freedom, *i.e.* we only fix some variables to arbitrary values and leave the rest untouched.

Example 3. Using the same notation as before, we have $K := \{a, b, c\}, V := \{\alpha, \beta, \gamma\}, F := \{\beta, \gamma\}$ and $f : K \cup V \to \mathbb{B}$. . Moreover, we consider $\tau : \{\alpha\} \to 0$. Note that we specialize the function from example 2 in the variable α and obtain

$$f(K, F) = a\beta + a\beta\gamma + b\beta\gamma + bc + a + b + \beta\gamma + 1$$

This leads to the following cubes:

cube	superpoly	degree	classification
$\emptyset$	$bc + a + b + 1$	2	quadratic
β	a	1	**linear**
$\beta\gamma$	$a + b + 1$	1	**linear**

These are clearly less cubes than in examples 1–2. By extending the assignment function τ to $\overline{\tau} : \alpha \to 0, \beta \to 1, \gamma \to 0$, we obtain the function f as

$$f(K, \overline{\tau}) = bc + b + 1$$

By cube-summing over β, we obtain the (linear) superpoly a. This is in line with the results given in example 1.

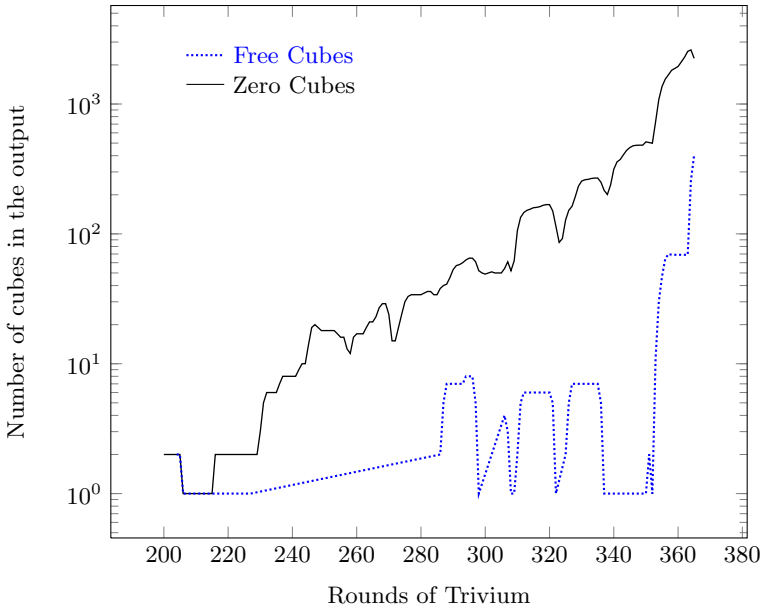


Fig. 1. Total number of zero and free cubes in the output of Trivium (rounds 200–365).

Fixing variables. Here we briefly discuss how fixing variables affects the residual polynomial $r(V, K)$. Let μ be a monomial in r with $|\mu| \geq 2$. Moreover, we have some variable $v_i \in \mu$. By setting $v_i = 0$ the monomial μ will vanish. In addition, if μ was the only monomial with $x_C|\mu$ in r this allows a new cube C. On the other hand, setting $v_i = 1$ leads to a new (restricted) monomial $\mu' := \mu\backslash\{v_i\}$. Assume that μ' *is* already part of the initial residual polynomial $r(V, K)$. In this case, the newly generated $\mu\backslash\{v_i\}$ will simply cancel out μ' and hence may foster the existence of another cube. So both may be useful to generate cubes for a given function f.

We want to stress that flexible cubes are a strictly weaker notion than free cubes: Fixing one additional variable in a free cube cannot destroy the cube-property. Any free cube is also a fixed cube for any assignment α over $V\backslash C$. Similarly, we can start with a flexible cube and fix up to $|F|$ variables until it becomes as fixed cube (or even a zero cube, if all variables become zero in α). This can be seen in figure 1 as the number of fixed cubes exceeds the number of free cubes, except for rare cases where both number are equal.

From a cryptanalytic point of view, fixed/zero cubes are more desirable than free cubes as they are more frequent than the latter. Consequently, we have a higher probability of finding them. However, if we want to combine other attacks with cube attacks, free cubes or maybe flexible cubes are more interesting as we only *take away* variables from the cube-set C (or S, respectively). All other variables are still available for some other attack, cf. section 2.6 in the case of dynamic cubes.

2.5 State cubes

Usually cubes are limited to the output of a cipher. This makes it impossible to use them in the initialisation phase of a cipher. Unfortunately, if we don't find cube equations in its output, the attack is useless against this cipher. Still, in practice, we can view the cipher as a white box and

consequently taking advantage of its internal structure. We try to capture this with the following definition.

In this section let $\mathcal{V}$ be the space of the possible IV's, $\mathcal{K}$ the key space and $\mathcal{S}$ the space of the states of a cipher. Denote $u : \mathcal{V} \times \mathcal{K} \times \mathcal{S} \to \mathcal{S}, (I, K, S) \mapsto S'$ the update funtion of the cipher. A cube in u or in a part of is called a *state cube*. They nicely relate to the Higher Order Differentials from *Lai* [Lai94].

The main goal of state cubes is to simplify the internal state of a cipher from an attacker's point of view. In order to do so we need an algebraic modelling technique for the cipher that allow us to handle many instances of the cipher and than add some state bits. With this in mind we can simplify the state of the cipher symbolically. We will discuss this in more length in Section 4. A concrete example of state cubes is given there as well, cf. example 9.

2.6 Dynamic Cubes

Dynamic cubes were first described in [DAS11] where they were used to break full Grain-128. Dynamic cubes can be seen as a special case of free cubes. Let $S \subset V$ be the set of all fixed variables in the original cube attack. We take a subset $D \subset S$, called the set of *dynamic variables*. They are used to define a function $d(V, K)$ in the public variables and some key variables. Each such function d is chosen in a way that some state bits cancel out. Alternatively, we can use the function d to get some information of the cipher that can be used by *cube testers*. Finding the function d requires a careful analysis of the internal structure of the cipher under attack. The following example is taken from [DAS11]. It illustrates the overall idea of dynamic cubes.

Example 4. We consider a polynomial P which is decomposed into $P = P_1P_2 + P_3$ where P_1, P_2 and P_3 are polynomials over $k_1, \ldots, k_5$ and $v_1, \ldots, v_5$ with

$$\begin{aligned} P_1 &:= v_2v_3k_1k_2k_3 + v_3v_4k_1k_3 + v_2k_1 + v_5k_1 + v_1 + v_2 + k_2 + k_3 + k_4 + k_5 + 1 \\ P_3 &:= v_1v_4k_3k_4 + v_2k_2k_3 + v_3k_1k_4 + v_4k_2k_4 + v_5k_3k_5k_1k_2k_4 + v_1 + k_2 + k_4 \end{aligned}$$

and P_2 is an arbitrary dense polynomial in the variables.

If we can force polynomial P_1 to zero than $P = P_3$. From the above definition we see that P_1 is a rather simple polynomial. First we set $v_4 = 0$ and make use of the linearity of v_1 in P_1 by setting $v_1 = v_2v_3k_1k_2k_3 + v_2k_1 + v_5k_1 + v_2 + k_2 + k_3 + k_4 + k_5 + 1$. This enforces $P_1 = 0$. During cube summation the value of v_1 will change. This is the difference between dynamic cubes and the definition of cubes as in Section 2.

Now we guess all values necessary to calculate v_1. In particular, these are $k_1, k_1k_2k_3$ and $k_2 + k_3 + k_4 + k_5 + 1$ (!). Plugging in the for v_1 and v_4 we get:

$$P = v_2v_3k_1k_2k_3 + v_2k_2k_3 + v_3k_1k_4 + v_5k_3k_5 + k_1k_2k_4 + v_2k_1 + v_2 + k_3 + k_5 + 1\,.$$

So we have a rather easy polynomial to attack with cubes of degree 2 in the IV variables and degree 3 in the key variables.

2.7 Mixed cubes

Until now, we have worked within the framework of cubes. Here, we will provide an alternative, but nevertheless useful definition from a cryptanalytic point of view. Let $O \subset V$ the *oracle variables* with $O \cap S = \emptyset$ and a function f of the form

$$f(V, K) = x_C p(K, O) + r(F, K)\,.$$

If $\forall \mu \in r : x_C \not| \mu$ we call C a *mixed cubes*.

At first glance, this definition is not very useful as it mixes IV variables into the key equations of the superpoly. However, at second glance this changes as we can now reduce the dimension of the cube C: As long as $x_C \nmid \mu$ holds for all $\mu \in r$, we can readily absorb variables from the set C into the set O and therefore reduce the cube dimension. In addition, we now have a much better source of (nonlinear) equations in the key variables: By setting the variables from O to different values, we obtain different (not linearly dependent) versions of the superpoly $p(K, O)$. Basically, we can "switch on" different monomials in $p(K, O)$ by setting the corresponding oracle variables to 1. Furthermore this new definition is less restrictive; consequently, we can expect more possible cubes and also cubes for a larger number of rounds than for the original definition of a cube.

To reflect this, we also need to change the definition of the *key-degree* of a mixed cube. It now becomes the maximal number of key-variables in any monomial in the superpoly $p(K, O)$. Formally:

$$\text{key-degree}(f) := \max_{\alpha: O \to \mathbb{B}} \{\deg(p(K, O)_{|\alpha})\}$$

Example 5. Consider $K := \{a, b, c\}, V := \{\alpha, \beta, \gamma\}$ be the sets of key and IV variables, respectively. In addition, we consider the oracle variables $\{\alpha, \beta\}$ and the function $f : K \cup V \to \mathbb{B}$ as

$$f(K, V) := 1 + a + c + ab + \alpha a + +\gamma a + \gamma b + \alpha\gamma + \alpha\gamma a + \beta\gamma b$$

This leads to the following mixed cubes:

cube	superpoly
$\emptyset$	$ab + a + c + 1 + \alpha a$
γ	$a + b + \alpha + \alpha a + \beta b$

We see that the only "real" cube is γ. In addition, the superpoly corresponds to different equations, depending on the choice for the IV variables α, β. In particular we have

(β, α)	superpoly
$(0, 0)$	$a + b + c$
$(0, 1)$	$b + c + 1$
$(1, 0)$	$a + c$
$(1, 1)$	$c + 1$

So using $p(K, (1, 1))$ we can compute the value of the key variable c. Furthermore, using $p(K, (0, 1))$ and $p(K, (1, 0))$ yields the values of of the key variable a, b.

Note that the corresponding cube summation does not take $2^3 = 8$ but only $3.2 = 6$ values of the underlying function f.

2.8 Polycubes

Until now, cubes were only defined for *cube monomials*. In this section, we consider a generalization. Let $\mathcal{C} = \{C_1, \ldots, C_k\}$ be a set of k cubes with the corresponding superpolys $p_1(K), \ldots, p_k(K)$. Consider their sum

$$s(\mathcal{C}) := \sum_{i=1} kp_i(K)\,.$$

We call $s(\mathcal{C})$ a *polycube*. In particular, for $\deg(s) = 1$, we have a *linear* polycube.

Example 6. Following the notation of the other examples so far, we investigate the function

$$f(K, V) := ab\alpha + ab\beta + a\beta + a\beta\gamma + b\beta\gamma + bc\alpha + bc\beta\gamma + \beta\gamma$$

Among others, this leads to the following cubes:

cube	superpoly
α	$ab + bc$
β	$ab + a$
$\beta\gamma$	$bc + a + b + 1$

Considering the polycube defined by $\mathcal{C} = \{\alpha, \beta, \beta\gamma\}$. This leads to the superpoly $s(\mathcal{C}) = b+1$. As its degree is 1 rather then 2 $\{\beta\gamma, \beta, \alpha\}$ it is more useful for cryptanalytic attacks than the individual cubes from $\mathcal{C}$.

While polycubes are much more general than the other cubes treated so far, they are also much harder to find. For example, the algorithm from Section 3 can easily detect fixed cubes and even flexible cubes. However, it cannot find polycubes. Hence we have to state it as an open problem to give an efficient method to extract meaningful polycubes from a given cryptographic primitive, in particular when it is given in partial ANF (see below). Note that polycubes are a generalization of the *cancellation attack* from [Vie08].

2.9 Factor cubes

As before, we ease the restriction that the cube must be a single monomial. Instead of having a *monomial*, we use a *polynomial* $\sigma(C)$. The defining equation of a cube now becomes

$$f(K, V) = \sigma(C)p(K) + r(V, K)\,.$$

In a sense, the term $\sigma(C)p(K)$ "factors out" $f(K, V) + r(V, K)$, hence the name of this cube. In addition, we require $\sum_{v \in I_C} \sigma(v) = 1$ and $\sum_{v \in I_C} r(v, V \backslash C, K) = 0$. Using the same argument as for theorem 1, the latter condition becomes $\nexists \mu \in r : x_C | \mu$ for a free factor cube and $x_C \notin r$ for a fixed factor cube.

While looking strange at first glance, this actually captures all required properties of a cube: When adding over all possible assignments $v \in I_C$, we have exactly one instance of the superpoly left. All others cancel out. In addition, the residual polynomial r will cancel out, too. Even more strange, we cannot tell apart the above definition from the initial definition when looking at *black box* cubes. From the behaviour of f, we simply cannot distinguish if it splits into a cube monomial x_C and a superpoly $p(K)$ and a residual polynomial r or cube polynomial $\sigma(C)$ and p, r. Unfortunately, when computing symbolically (*e.g.* Section 5), factor cubes are much harder to detect than cubes based on monomials. In any case, such factor cubes exist as we see in the following example.

Example 7. As before, we have $K := \{a, b, c\}$ and $V := \{\alpha, \beta, \gamma\}$. Moreover, we consider the function

$$f(K, V) := (\alpha + \beta + \alpha\beta)(a + b + 1) + (\alpha + \beta + \gamma)(a + b + c)$$

Using the definition from above we obtain the cube polynomial $\sigma(\{\alpha, \beta\}) = \alpha + \beta + \alpha\beta$, the superpoly $p(K) = a + b + 1$ and the remainder $r(K, V) = (\alpha + \beta + \gamma)(a + b + c)$.

3 Trivium

To demonstrate our framework we use the stream cipher Trivium. It is a well-known hardware oriented synchronous stream cipher presented in [CP08]. Trivium generates up to 2^{64} keystream bits from an 80 bit IV and an 80 bit key. The cipher consists of an initialisation or "clocking" phase of R rounds and a keystream generation phase. There are several ways to describe Trivium—below we use the most compact one with three quadratic, recursive equations for the state bits and one linear equation to generate the output. The two only operations in Trivium are addition and multiplication over GF(2) as this can be implemented extremely efficient in hardware.

Consider three shift registers $A := (a_i, \ldots, a_{i-92}), B := (b_i, \ldots, b_{i-83})$ and $C := (c_i, \ldots, c_{i-110})$. They are called the *state* of Trivium. The state is initialized with $A = (k_0, \ldots, k_{79}, 0, \ldots, 0), B = (v_0, \ldots, v_{79}, 0, \ldots, 0)$ and $C = (0, \ldots, 0, 1, 1, 1)$. Here $(k_0, \ldots, k_{79}) \in \mathbb{B}$ is the *key* and $(v_0, \ldots, v_{79}) \in \mathbb{B}$ is the *initialization vector (IV)* of Trivium. Recovering the first vector is the prime aim of attackers. Note that the second vector is actually known to the attacker. In cube attacks, we even make the assumption that an attacker can fully control the IV used within the cipher and obtain a stream of output bits for a fixed key and any choice of initialization vector (IV).

The state is updated using to the following recursive definition.

$$\begin{aligned} b_i &:= a_{i-65} + a_{i-92} + a_{i-90}a_{i-91} + b_{i-77} \\ c_i &:= b_{i-68} + b_{i-83} + b_{i-81}b_{i-82} + c_{i-86} \\ a_i &:= c_{i-65} + c_{i-110} + c_{i-108}c_{i-109} + a_{i-68} \end{aligned}$$

After a clocking phase of R rounds, we additionally produce one bit of output using the function

$$z_i := c_{i-65} + c_{i-110} + a_{i-65} + a_{i-92} + b_{i-68} + b_{i-83}\,.$$

Note that the three state update functions correspond to 3 linear feedback shift registers with 5 tap positions each. Each polynomial is irreducible over GF(2) and hence provides an LFSR of maximal period. In each LFSR, the second-last term has been replace by a quadratic monomial. This is the only non-linear component in Trivum. Our results are based on round-reduced versions of Trivium with special choices of R. Full Trivium uses $R = 1152$ initialization rounds. Until now, no successful attack is known against full round Trivium.

There exist further variants of Trivium using two instead of three shift registers [CP08]. They are called Bivium-A and Bivium-B.

3.1 Brief overview of attacks on Trivium and round-reduced Trivium

We give a brief sketch some of the most important attacks against Trivium. Our aim is to highlight that Trivium is still secure—despite its simple and elegant design; and the combined effort of the cryptanalytic community.

The attacks from [DS09a, FV13] are both cube attacks. They recover the full key of a 799 round-reduced variant of Trivium in 2^{62} computations. Nevertheless, more attacks on Trivium are known so far. In [KMNP11, ADMS09, Sta10] their are some distinguishers attacks based on cube testers that cover up to 961 rounds but only work in a reduced key space of 2^{26} keys. Time complexity is 2^{25} computations.

Pure algebraic attacks are given in [SFP08, T$^+$13, SR12, Rad06]. They break the both reduced variants Bivium-A and Bivium-B. They fail for Trivium, even in its round-reduced version. Main reason is that they all begin their attack after the initialization phase so they are limited to one instance of Trivium.

Other attacks are not as succesful. For instance [KHK06] contains a linear attack on Trivium breaking a 288 round-reduced variant with a likelihood of 2^{-72}.

4 Similar variables and state cubes

The cube attack as defined in [DS09a] views the corresponding cipher as a black box. So there is no chance to take the initialisation phase of a given cipher into account. Thus cubes are limited to the output of the cipher; consequently, if we do find cubes for the output of the full cipher the attack fails.

By Kerckhoff's principle, we actually know which cipher we attacking, so we can overcome this disadvantage by building a suitable algebraic model. Here we do this exemplary on the stream cipher Trivium. However, the presented modelling technique is by no means limited to this cipher.

Algebraic attacks in [SFP08, T$^+$13, SR12] are based on the same algebraic representation of Trivium, namely the one given in [Rad06]. So all state bits of one Trivium instance are set to symbolic variables after the initilization phase. In the first model they intoduce three new variables for a_i, b_i and c_i every time the output is generated; we denote the number of output bits by n_o. Based on the above defintions, they obtain a sparse quadratic system of equations with $288 + 3 * n_o$ variables and $4 * n_o$ equations . In the second model the authors of [Rad06] do not introduce any intermediate variables. Therefore the equations are of ascending degree in the 288 state bits and the system of equations becomes dense. Consequently, it is also much harder to solve. All in all these strategies have limited success as they broke Bivium-A and Bivium-B but did not manage to break Trivium or even round-reduced versions of Trivium.

Our goal is to use cubes which occur in the state bits of Trivium. Therefore we need an algebraic representation that can both handle more than one Trivium instance and is also able to use information from the internal state.

To this aim, we only use symbolic values of the key variables. Then we update Trivium a number R of rounds only generating intermediate variables when we need them to bound the degree of the equations by two. This allows us to generate many Trivium instances with the same key but with different IV vectors.

The downside of our strategy is the growing number of intermediate variables. Therefore, we define the relation of *similar variables* and use linear algebra techniques to minimize the overall number of variables.

4.1 Algebraic representation using similar variables

Let $I \subset V$ be a subset of the IV variables. We consider the first n_o output bits of Trivium instances that are all defined by the same key and all vectors from I_S for some fixed set S, cf. the definition of I_S in section 2.1.

In our approach we set up all Trivium instances with symbolic variables $k_0, \ldots, k_{79}$ for the key and set the IV variables corresponding to vectors in $\mathbb{B}^{|S|}$. Denote the current Trivium instance by $t \in \mathbb{N}$. We initialize these instances for a given number of rounds R and introduce three new variables every round for the entries $a_{t,i}, b_{t,i}$ and $c_{t,i}$ in the three Registers A_t, B_t and C_t. This produces a quadratic system with a large amount of variables and monomials. Therefore we introduce some methods to reduce the number of variables. This reduction of the number of variables and monomials is important because solving techniques such as Gröbner-bases algorithms depends heavily on the number of variables. First of all we consider one Trivium instance in the following lemma.

Lemma 1. *Let $R > 238$ and $n_o \leqslant 66$. With the algebraic modelling technique described above we use exactly $3R - 522$ intermediate variables to describe one Trivium instance.*

Proof. We want to count the intermediate variables generated while modelling one Trivium instance. To do so we consider the update function

$$\begin{aligned} b_i &= a_{i-65} + a_{i-92} + a_{i-90}a_{i-91} + b_{i-77} \\ c_i &= b_{i-68} + b_{i-83} + b_{i-81}b_{i-82} + c_{i-86} \\ a_i &= c_{i-65} + c_{i-110} + c_{i-108}c_{i-109} + a_{i-68} \end{aligned}$$

of Trivium. Whenever we would get an equation of total degree greater than two we set the quadratic equation to a new intermediate variable and continue the calculation with it. Our proof deals with each register A, B, C one after the other.

At the begining register A is the only one containing symbolic values. In the 13th round the first quadratic expression $b_{12} = a_{78} \cdot a_{79} + \cdots = k_{79} \cdot k_{78} + \cdots$ is produced. Inspecting the update equations of Trivium, we see that it takes 82 rounds until it will be multiplied in round 94 with a linear element in c_{95} and the first intermediate variable will be introduced. After round 94 we use one new variable in the register C every round.

We count the intermediate variables in the other registers in an analogous fashion. First we consider register A. The above mentioned quadratic expression b_{12} will also be stored in register C in round $13 + 69 = 82$ because of $c_{80} = b_{12} + \cdots$. In round 191 this expression will be multiplied with an linear expression in $a_{189} = c_{81} \cdot c_{80} + \cdots$ and we have to introduce a new intermediate variable in the register A. Thus after round 190 there will be a new variable in register A and C in each round.

Now we investigate the register B. As mentioned above our quadratic expression b_{12} will stored in register C in round 82. After 66 more rounds it is stored in $a_{145} = c_{80} + \cdots$. From there it takes further 91 rounds until a new variable is required in $b_{236} = a_{146} \cdot a_{145} + \cdots$. So starting in round 239 we will need a new intermediate variable in every register without further reduction techniques.

So all in all, we have the following numbers of intermediate variables ν:

$$\begin{aligned} \nu &= (R - 94) + (R - 190) + (R - 238) \\ &= 3R - 522. \end{aligned}$$

Finally we note that we do not have to introduce new variables while producing the first 66 rounds of output. After the initialization phase we are just interested in output equations. The

output function of Trivium

$$z_i = c_{i-65} + c_{i-110} + a_{i-65} + a_{i-92} + b_{i-68} + b_{i-83}$$

is linear and uses at most 66 rounds old state bits. Thus we do not have to clock the registers and so do not introduce new intermediate varibales if we want $n_o \leqslant 66$ output equations. □

Corollary 1. *For $0 \leqslant n_o \leqslant 66$ output bits, we need to compute only $R + n_o - 66$ updates of the state in Trivium.*

For the remainder of this section we assume $R > 238$. Now we take a more general point of view and introduce similar variables for generalized systems of equations. Nevertheless we will later on use this observation for a system of *many* instances of Trivium.

Definition 1. *Let $\mathbb{R} = \mathbb{F}_2[k_0, \ldots, k_{79}, y_0, y_1, \ldots] =: \mathbb{F}_2[K, Y]$ be the Boolean Polynomial Ring in the key Variables K and all intermediate variables Y.*

We call two intermediate variables y_i and y_j similar iff $y_i + y_j = p(K, Y\backslash\{y_i, y_j\})$ where $p(K, Y\backslash\{y_i, y_j\})$ is a polynomial of degree $\deg(p) \leqslant 1$.

Taking similar variables into account we can save many intermediate variables. Whenever we want to introduce a new intermediate variable we first test if there exists a similar one. If yes, do not introduce a new one but use $y_j + p(K, Y\backslash\{y_j\})$ instead.

Furthermore if we have the set F of polynomial equations in $\mathbb{R}$ introducing the intermediate variables, the so-called set of system equations, we can generalize the definition above as follows:

Definition 2. *Let $\mathbb{R} = \mathbb{F}_2[k_0, \ldots, k_{79}, y_0, y_1, \ldots] =: \mathbb{F}_2[K, Y]$ be the* Boolean Polynomial Ring *in the key variables K and all intermediate variables Y. Denote with F the* system equations*.*

Denote by $Y_k \subset Y, Y_k \neq \emptyset$ for $1 \leq k < |Y|$ the sets of already introduced intermediate variables up to step k. We call the intermediate variable y_i similar *to the set F iff there exist a non-empty set Y_k such that $y_i + \sum_{y \in Y_k} y = p(K, (Y\backslash Y_k)\backslash\{y_i\})$ where $p(K, (Y\backslash Y_k)\backslash\{y_i\})$ is a polynomial of degree $\deg(p) \leqslant 1$.*

The following example illustrates how we work with similar variables.

Example 8. Consider the equations defining intermediate variables

$$\begin{aligned} y_0 &= k_{78}k_{79} + k_{53} \\ y_1 &= k_{77}k_{78} + k_{79} + k_{52} \end{aligned}$$

in the system of equations F. Assume we want to introduce the new intermediate variable

$$y_2 = k_{79}k_{78} + k_{78}k_{77} + k_5 + k_{61}.$$

We have $y_2 = y_1 + y_0 + k_{53} + k_{79} + k_{52} + k_5 + k_{61}$. So we do not need y_2 and we can go on computing with y_0 and y_1 instead. This does not only save us one variable but we also have replaced a quadratic equation by a (potentially more useful) linear one.

So we can save a lot of intermediate variables using similar variables. This is particularly true when generating more than only one Trivium instance. Table 1 shows some experimental results on modelling Trivium for a reduced number of rounds. Here, we have generated 32 instances of Trivium with 66 output bits.

Table 1. R: Number of initial rounds, $3R-522$: One instance without similar variables, ν: number of intermediate variables; the number of output bits is set to 66 and 32 Trivium instances are generated.

R	$3R-522$	ν
400	678	548
450	828	961
500	978	1489
550	1128	3510
600	1278	6557

The first column of Table 1 gives the number of initialization rounds for each Trivium instance. In the second one we have the number of variables used for one Trivium instance without similar variables and the number of intermediate variables which are used to describe the whole system is ν. We see that have greatly decreased the number of variables. Usually we would need $1278*32 = 40896$ variables to model 32 instances 600-round-reduced version of Trivium. We also have produced $66 * 32 = 2112$ output equations. The model in [Rad06] can just handle one instance and needs $288 + 3 * 2112 = 6624$ variables to produce that amount of output equations. In our model we see a saturation of the number of variables and monomials. We discuss this point in more depth later in this article. Thus increasing the number of instances actually increases our advantage, too.

These systems are out of reach for nowadays Gröbner bases implementations like PolyBoRi [BD09]. The number of variables is simply too high.

We describe algorithm 1. It is used to generate the model outlined above, but is specialized to Trivium. Algorithm 1 returns a system of equations F that represent $T \in \mathbb{N}$ instances of Trivium with R rounds and $n_o \leqslant 66$ output. The key variables $k_0, \ldots, k_{79}$ are shared among these instances. However, the IV variables $v_{t,0}, \ldots, v_{t,79}$ are initialized with different values for each individual instance. In the function representation the three shift registers are initialized and output is produced according to the function `updateState` for the t instances of Trivium. After setting up the system F we echelonize it by interpreting it as a matrix with monomials as columns and polynomials as rows. The echelonized form creates two sets, namely quadratic equations Q and linear equations L. Using the linear terms from L, we can simplify Q. To this aim, we replace the leading terms $\mathrm{LT}(L)$ of L in Q by their corresponding equations. In doing so we use *Degree Reverse Lexicographic* ordering with a round ascending ordering and put key variables first. In this case, echelonizing and plugging in $\mathrm{LT}(L)$ in Q directly makes use of similar variables. This follows as the echelonization algorithm first puts intermediate variables of higher rounds before those of lower rounds. All linear relations are then used to produce even more similar variables. `insertLinVars` is plugging these linear equations into the rest of the equation system.

Before discussing the running time of the algorithm 1 we note that it can be applied to other ciphers as well. When other ciphers have a quadratic update function, we can directly apply algorithm 1. Otherwise, we need to take care of the higher degree of the update function.

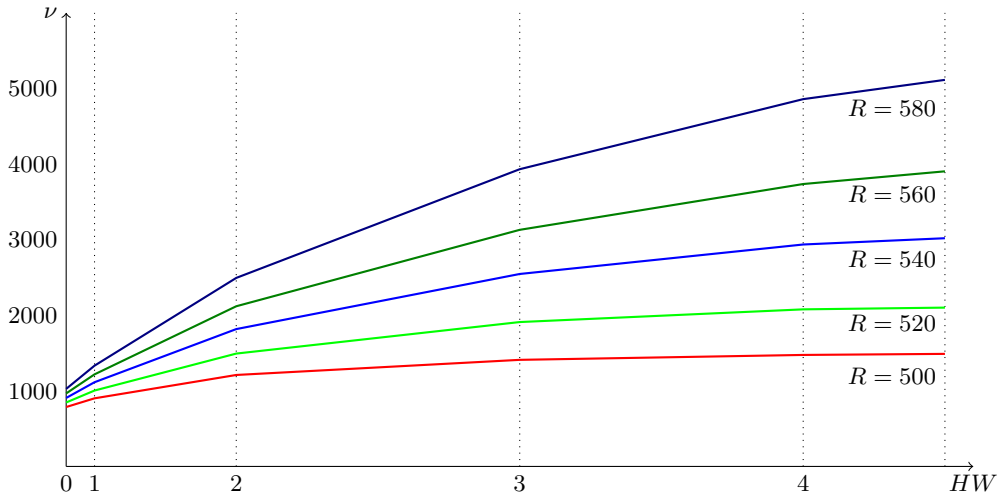
To evaluate the running time of algorithm 1 we consider the different parts of it. First of all we take a look at the generation of one Trivium instance. The state will be updated $R+n_o$ times. Each time there are 3 multiplications of linear polynomials. So it has time complexity $\mathcal{O}((R+n_o) \cdot n_l^2)$ where n_l is the maximal number of terms used in a linear polynomial.

Algorithm 1 Generating the algebraic representation of Trivium F using similar variables. The function echelonize echelonize the matrix coresponding to F and insertLinVars plugs the linear variables into the system F

function updateState(k, out):
if out = **true then**
    F.insert($c_{i-65} + c_{i-110} + a_{i-65} + a_{i-92} + b_{i-68} + b_{i-83}$)
    return
end if
F.insert($a_{t,k} + A[0]$); F.insert($b_{t,k} + B[0]$); F.insert($c_{t,k} + C[0]$)
$A[0] \leftarrow a_{t,k}$; $B[0] \leftarrow b_{t,k}$; $C[0] \leftarrow c_{t,k}$
for $j \leftarrow 0$ **to** 91 **do**
    A[j+1] = A[j]
end for
A[0] = $c_{i-65} + c_{i-110} + c_{i-108}c_{i-109} + a_{i-68}$
for $j \leftarrow 0$ **to** 82 **do**
    B[j+1] = B[j]
end for
B[0] = $a_{i-65} + a_{i-92} + a_{i-90}a_{i-91} + b_{i-77}$
for $j \leftarrow 0$ **to** 109 **do**
    C[j+1] = C[j]
end for
C[0] = $b_{i-68} + b_{i-83} + b_{i-81}b_{i-82} + c_{i-86}$
return

function representation(R, n_o, t):
global $F \leftarrow \emptyset$
for $n \leftarrow 1$ **to** t **do**
    global $A \leftarrow (k_0, \ldots, k_{79}, 0, \ldots, 0)$; $B \leftarrow (v_{t,0}, \ldots, v_{t,79}, 0, \ldots, 0)$; $C \leftarrow (0, \ldots, 0, 1, 1, 1)$
    for $k \leftarrow 0$ **to** $R - 1$ **do**
        updateState($k, out =$ **false**)
    end for
    for $k \leftarrow 1$ **to** n_o **do**
        updateState($k, out =$ **true**)
    end for
    echelonize(F)
    insertLinVars(F)
end for
return F

Fig. 2. Saturation of variables; Hamming Weigth against number of variables for R rounds of Trivium



The matrix corresponding to F is an $m \times n-$matrix where n is the number of monomials and m is the number of equations in F. In the update function there will be at most 3 new equations per round. After 111 rounds these polynomials consists of 4 monomials each. So there are more monomials than equations since we have $R > 238$. As a consequence the running time of step 2 can be approximated by $\mathcal{O}(n^3)$. Note that n grows per instance we generate. As we will see below, the number of variables and monomials saturates.

The function `insertLinVars` has a running time of $\mathcal{O}(n_p^2 \cdot m)$. We have to insert all m equations of the system to at most the maximum number of monomials n_p in *one* equation in F.

We get an overall running time of $\mathcal{O}(Tn^3)$ since the echelonization is the most expensive step and we perform it at most t times.

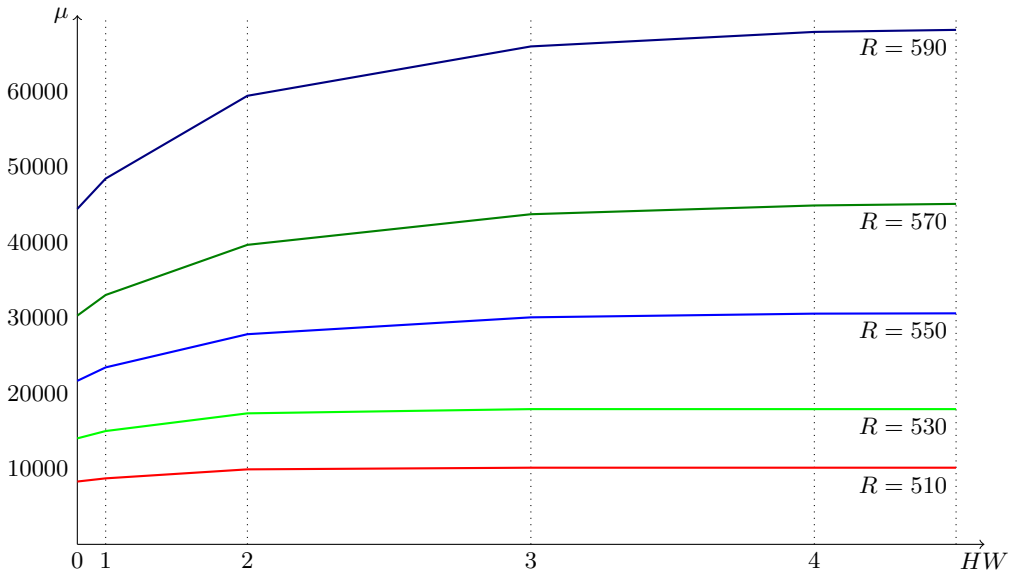
In practice, this number will be lower. For example, we can use an implemenation of Strassen's algorithm So the running time is more likely $\mathcal{O}(Tn^{2.7})$ or $\mathcal{O}(Tn^2)$ as long as the system F is sparse.

Saturation. We can see in figures 2–3 that similar variables greatly effect the number of variables. In our experiments we found a saturation of the number of variables and monomials that depends on the Hamming weight of the IV we are using and on the initialization rounds R.

When generating many instances we take i IV bits and produce the 2^i instances corresponding to all possible vectors from $\mathbb{B}^i$. When generating an instance of high Hamming weight we spend less variables and monomials than for an instance of low Hamming Weight. When the number of rounds grows we need to generate more instances to see this effect. We have plotted this effect for 32 instances in figures 2–3, counting both the number of variables and the number of quadratic monomials.

Figure 2 shows the Hamming weight of the IV against the number of variables ν, starting at round 500 up to round 580 in steps of 20. Instances with the same Hamming weight yield the same amount of variables. This amount is lowered when we generate instances with higher Hamming weight. The saturation of monomials needs a lower Hamming weight of the IV as can be seen in figure 3. Here, we have started at round 510 and went up to round 590 in steps of 20. It shows

Fig. 3. Saturation of monomials; Hamming Weigth against number of quadratic monomials for R rounds in Trivium



Hamming weigth against the number of quadratic monomials μ in F. All in all, the saturation of monomials is much flatter then the saturation of variables. Note that variables that are not saturated are found in the linear terms.

This means that at some point we get more system defining equations from the output than new variables or monomials. So our system gets defined or even overdetermined when we have enough instances. Note that we did not take the output equations into account yet but only the defining system itself. The output introduces additional, unknown monomials but no new variables.

4.2 Taking advantage of the white box

With the modelling techniques described above we are able to obtain the symbolic values for any state bit in any round from any instance of Trivium. This allows us to (cube) sum certain state bits from different instances to form state cubes (see section 2.5).

First we note that similar variables generate cross references between all instances we produce. Second, we directly obtain linear equations in the key variables and use them to obtain more similar variables. Finally, if the number of rounds is low enough, we even get linear equations between the output and key variables. In this case, we can simplify the system even further.

Now we have access to a full symbolic state so we can add state bits from many instances. This is one assumption of the cube attack. The second one is the existence of cubes in the state. How to find cubes is stated in section 5. With these techniques we found many state cubes. Detailed results are given in section 6.

While cross references coming from similar variables are a good thing, there also is a catch. In particular, they annihilate state cubes as the corresponding equations will be generated by similar variables, too. So we can only use state cubes from higher rounds. Still, the advantage using state cubes is that we can use *all* of them as they all introduce linear relations between instances. Moreover, there is no need to find cubes in the output which is the case for the native cube attack.

For reference: Until round 699 we have found about $31,000$ state cubes within the first 12 IV variables. We have used the algorithm from Section 5.4 for this search. Note that state cubes do not lead to a linear system in the key bits. They establish more cross references between the key bits and the state bits from initialization rounds. So when we have assembled enough state cubes in the system F we skip even further initialization rounds. When we have a full representation of the cipher using algorithm 1 we can use it to generate additional equations from known state cubes.

Example 9. To see that state cubes actually exist, consider

state/round	Cube	superpoly
b_{302}	v_0	k_{65}
a_{350}	v_{11}	k_{15}
b_{546}	$v_0v_1v_3v_9$	$k_{55}+k_{61}$
c_{586}	$v_0v_1v_2v_3v_4v_5v_7v_8v_9v_{11}$	k_9
a_{601}	$v_0v_2v_3v_4v_5v_6v_7v_8v_9v_{11}$	k_{65}

Also these examples are completely arbitrary taken from the list of possible state cubes, they can be used to illustrate the idea: Take b_{302}. This leads to the equation

$$b_{0,302}+b_{1,302}+k_{65}=0$$

that is valid independent of the choice of k_{65}. By replacing $b_{0,302}$ and $b_{1,302}$ with their corresponding representation in k_i, y_j, we obtain a new equation that we can add to the overall system F.

5 Cube finding

The main technique for finding cubes in a given cipher is *black box polynomial testing*. This means we view the corresponding function as a black box and try to interfere a (partial) ANF and consequently cube equations. Most successful for this purpose are a random-walk algorithm [DS09a] and its successor [FV13]. The latter uses additional memory to speed up the computation. For the sake of simplicity, we only state the former for the case of linear cubes.

Random Walk Algorithm. We start with a random set $C \subset V$ and compute the superpoly $p(K)$ by multivariate polynomial interpolation. There are three possible outcomes: the degree of $p(K)$ is 1. In this case, we output the pair (C,p) as a new cube/superpoly. Alternatively, the degree may be zero or negative. In this case, the set C was too large and we drop one element from it. If we cannot compute a linear superpoly, its degree is higher than 1 and we need to add another variable to the set C. If we do not succeed after a certain number of trials, we restart with a fresh set C.

Unfortunately, the analysis of both algorithms is rather tricky, so we do not know how many / which type of cubes they will find. Consequently, we cannot make statements like "Trivium does not have cubes of dimension k in round r" but only "Our algorithm did not find anything (yet)". In particular, there is quite some freedom in choosing the "correct" starting set C and also to include / drop certain variables which further complicates the situation. This is obviously an obstacle to develop a full theory of cubes and their existence in cryptographic algorithms.

In addition and to verify the output of the random walk algorithms from above, a probabilistic linearity test such as [BLR93, JPRZ09] is used. When it outputs *not linear*, this is correct. In the other case, the output may be wrong with probability $2^{-\gamma}$ for some constant $\gamma \in \mathbb{N}$. In practice, we have $\gamma = 100$ which is reasonably certain for cryptographic applications. To the knowledge of the authors, there is no deterministic algorithm known to solve this problem.

5.1 Polynomial degree testing

As linearity testing and its generalization to higher degree polynomials are an important building block, we state the degree testing algorithm from [BLR93] both in its generalized form and in its (easier) linearized form. We start with the latter. Let $f(x) : GF(2)^n \to GF(2)$ be a Boolean function. We want to establish with high probability if f is linear, *i.e.* (1) holds.

$$\forall x, y \in \mathrm{GF}(2)^n : f(x+y) = f(x) + f(y) + f(0) \tag{1}$$

If the function f is available in full ANF, this is easy. However, if f is only available in black-box-access, the problem becomes difficult. A possible way out is the one-sided BLR linearity test [BLR93]. Here, we evaluate (1) for several, randomly chosen pairs $(x, y) \in_R \mathrm{GF}(2)^n \times \mathrm{GF}(2)^n$. If (1) is violated by at least one pair, we conclude that f is *not* linear. To confirm that f is indeed linear with probability 2^{-s} for some certainty level $s \in \mathbb{N}$, we need a total of $\lceil (4 + 1/4)s \rceil$ repetitions.

Algorithm 2 Compute the degree of a given function f. The degree must be within the set of target degrees D. Otherwise, return $\perp$

```
 1: function addSet(X):
 2: if |X| = 0 then
 3:     return 0
 4: end if
 5: a = 0
 6: for all x ∈ X do
 7:     a ← a + x
 8: end for
 9: return a
10:
11: function computeDegree(f, D, s)
12: d ← max(D); r ← 0
13: while r < ⌈(d2^d + 1/2^{d+1}) s⌉ do
14:     x_1, ..., x_{d+1} ∈_R GF(2)^n
15:     A = (...) {Empty sequence}
16:     for all σ ⊂ {1, ..., d + 1} do
17:         A_σ ← f( addSet({x_i : i ∈ σ}))
18:     end for
19:     D' ← ∅
20:     {Iterate over all potential degrees}
21:     for all δ ∈ D do
22:         {Iterate over all subsets of length δ}
23:         for all s ∈ {b ⊂ {1, ..., d + 1} : |b| = δ} do
24:             if addSet({A_σ : σ ∈ s}) ≠ 0 then
25:                 D' ← D' ∪ {δ} {We found a witness against degree δ}
26:             end if
27:         end for
28:     end for
29:     D ← D\D'
30:     if |D| = 0 then
31:         return ⊥
32:     end if
33:     d ← max(D); r ← r + 1
34: end while
35: return min(D)
```

In a more general framework, *i.e.* assuming that we want to confirm that f has degree d with probability 2^{-s}, we need a total of

$$\left(d2^d + \frac{1}{2^{d+1}}\right) s$$

repetitions of the above test. The corresponding lemma and algorithm are given in [JPRZ09]. Instead of stating it here, we further generalize it from degree *testing* to degree *detection*, that is, given a function f we determine its degree within a given range. See algorithm 2 for the pseudo-code. In particular, linear and quadratic functions (so $d = 1, 2$) are of high interest to us as they correspond to linear and quadratic cubes. Although such an algorithm seems to follow directly from the one given in [JPRZ09], there are some subtilities that allow a considerable speed-up in practice. In particular, we can save on (expensive) evaluations of the underlying function f by re-arranging them into different subsets (lines 23–27). This is possible as all values are chosen pairwise independent.

All in all, the algorithm samples $\left(\frac{1}{d2^d+2^{d+1}}\right) s(d+1)$ random vectors from the vector space $\mathrm{GF}(2)^n$ and needs $\left(\left(d2^d + \frac{1}{2^{d+1}}\right) s\right)2^{d+1} = \mathcal{O}(ds2^{2d+1})$ evaluations of the underlying function f. Consequently, it is not suitable to extract the degree of a large number of quadratic or even cubic cubes. As we will see in Section 5.5, this function is the real bottleneck when searching for a large number of cubes. So one of our goals is to develop more efficient filters to limit the use of the above algorithm to the absolute minimum. On the up side, the linearity tester can readily be parallellized as we can distribute different cubes to different machines with only minimal communication overhead.

5.2 Basic algorithm

In any case, when looking for cubes we are searching a needle in a hey-stack. Even worse, when using one of the random walk algorithms, we push the search in a certain direction. Therefore, "real" cubes might have different properties than the ones we find by these algorithms. To overcome this problem, we will discuss several algorithms that capture cubes more accurately. In particular, we have a tighter control over the probability space they operate on, so we can make better funded claims, *e.g.* about the distribution of cubes within Trivium.

The first algorithm is rather trivial, but nevertheless useful—both for Trivium up to round 446 (cf. Section 5.5) and also as a building block for further algorithms. In a nutshell, we use the full ANF of the corresponding function and then apply the corresponding cube criterion from Section 2. Although we could use any of the above criteria to determine the existence of a cube, we have specialized the algorithm to the case of linear zero (fixed) cubes as they have the highest relevance for direct cryptographic attacks. In addition, this eases explanation of the general idea.

The subfunction `split` is used to separate key variables from IV variables for a given monomial μ. Based on this, we add up all superpolys in lines 5–9. By definition, we only allow superpolys of degree 1 or lower. These are extracted in line 11.

Correctness. The correctness of the algorithm follows from the definition of fixed linear cubes: Starting with the ANF of f, we regroup all monomials by their IV variables. Adding all key-variables for a given x_C yields the corresponding superpoly (line 7). Excluding the one-monomial $\emptyset$ (line 10), we just need to test which potential superpoly actually has the correct degree (line 11).

The algorithm is illustrated in the following example.

Algorithm 3 Directly extracting cubes from a function f, given in Algebraic Normal Form

```
1: function split(μ) return (μ ∩ V, μ ∩ K)
2:
3: function extractCubes(f)
4: superPolys ← (0, ..., 0); potentialCubes ← ∅
5: for all μ ∈ f do
6:     ν, κ ← split(μ)
7:     superPolys_ν ← superPolys_ν + κ
8:     potentialCubes.insert(ν)
9: end for
10: potentialCubes ← potentialCubes \{∅}
11: allCubes ← {ν ∈ potentialCubes : superPolys_ν.degree() = 1}
12: return allCubes
```

Example 10. Let $K := \{a, b, c\}, V := \{\alpha, \beta, \gamma\}$ be the sets of key and IV variables, respectively. Moreover, consider the following function $f : K \cup V \to \mathbb{B}$ in algebraic normal form (ANF):

$$f(K, V) := ab\alpha + a\alpha\beta + a\beta + a\beta\gamma + b\beta\gamma + c\alpha\beta\gamma + bc\alpha + abc\alpha\beta + abc\alpha\gamma + bc + a + b + \beta\gamma + 1$$

Using algorithm 3, we have the following superpolys, indexed by IV sets:

IV set ν	superPolys$_\nu$	linear
$\emptyset$	$bc + a + b + 1$	-
α	$ab + bc$	-
β	a	$\star$
$\alpha\beta$	$a + abc$	-
$\alpha\gamma$	abc	-
$\beta\gamma$	$a + b + 1$	$\star$
$\alpha\beta\gamma$	c	$\star$

So consequently, our algorithm returns the set $\{\beta, \beta\gamma, \alpha\beta\gamma\}$. Although not a linear superpoly, $bc + a + 1$ is interesting, too. We note that the corresponding cube $\nu = \emptyset$; by definition, this corresponds to the monomial 1. It will occur in the cube summation of *every* other cube. In our case, this is the ANF of the restricted function $\overline{f}(K, (0, 0, 0)) = bc + a + 1$. In practice, this special form of the function f will exist for any given cipher, but has a far too high algebraic degree to be of practical use. Therefore, we rely on the fact that $\overline{f}(K, (0, 0, 0))$ is summed an even number of times and hence cancels out in the cube sum.

Running time. Its running time is bounded from above by the number of monomials in f and the length of the longest monomial μ in f. Denote these with $|f|$ and $|\mu|$. Then both the running time and the memory consumption are in $\mathcal{O}(|f||\mu|)$.

To adapt this algorithm to different variants of cubes (cf. Section 2), we can use small modifications. For example, to deal with higher order cubes, we change the degree condition in line 11; to allow for free cubes instead of fixed cubes, we need to verify that a potential cube actually contains *all* occurrences of its IV variables in the given ANF, *i.e.* that it is pairwise disjoint with all other potential cubes. The latter will slightly increase the running time but is still feasible in practice.

5.3 Pushing the limit

Up to now, we were dealing with the full ANF of the target function f. For cryptographic algorithms, this becomes prohibitively expensive as the degree of f (and also the number of monomials) usually increases exponentially, cf. section 6.1 and in particular figure 4 for the case of Trivium. Note that this is primarily a concern of the available memory, not so much of the available running time.

A simple but effective trick to reduce memory consumption is to evaluate f in some IV variables by simply setting them to zero. We denoted the result by $\overline{f}$. So instead of using algorithm 3 with the full function f, we use $\overline{f}$ as its input. In the remainder of this section we investigate the implications of this approach.

First, this idea has some similarities to the random walk algorithm. Here, we start at some random point and evaluate f "around" it. But it also has some advantages over the latter. In particular, our restriction is independent from the search algorithm so we can better control it. Consequently, we can easily estimate the number of cubes that *should* live at the evaluation of f in a given round, cf. lemma 2. Running the algorithm several times and combining the observations of several such restrictions increases the accuracy of the prediction. Note that this algorithm is particularly well suited for fixed cubes. Considering both the search space and the overall space of IV variables leads to the following lemma:

Lemma 2. *Let R be the number of cubes of dimension $r \geq 1$ in a function $f(K, V)$. Let $\overline{f}$ be its restriction such that we have $\overline{f}(K, \overline{V})$ for $\overline{V} \subset V$. Then we expect on average a total of*

$$R \cdot (|\overline{V}|/|V|)^r$$

cubes of dimension r in $\overline{f}$.

Proof. For the proof consider some cube C of dimension r and a randomly drawn set $\overline{V} \subset V$. For each variable in C we have a probability of $|\overline{V}|/|V|$ that it is also contained in the set $\overline{V}$. As all these r events are independent, the lemma follows for one cube and consequently for all R cubes in V.

However, to succeed in practice, we cannot compute the function f and then restrict it. This is impossible to the memory constraints. But if f is recursively defined, we can do it the other way around: First assign values to all corresponding variables and *then* develop f following its definition. This will lead to the same function $\overline{f}$. Note that such a recursive definition exists for Trivium in particular and also for all relevant cryptographic functions. For example for the AES, this was made explicit in [MR02, CMR05]. Similar work has been done for Present [Lea10] and other cryptographic primitives such as SHA-3; for this, we can use the so-called "Keccak-Tools", to generate the corresponding equations explicitly, including round reduced SHA-3.

Example 11. Using the same function f as in example 10, we choose $\overline{V} = \{\beta, \gamma\}$. Consequently, the restricted function $\overline{f}$ becomes:

$$f(K, V) := a\beta + a\beta\gamma + b\beta\gamma + bc + a + b + 1 + \beta\gamma$$

Using algorithm 3, we have the following superpolys, indexed by IV sets:

IV set ν	superPolys$_\nu$	linear
$\emptyset$	$bc + a + b + 1$	-
β	a	$\star$
$\beta\gamma$	$a + b + 1$	$\star$

As expected, the algorithm cannot identify all cubes anymore and the resulting set becomes now $\{\beta, \beta\gamma\}$. By also considering $\overline{V_1} = \{\alpha, \beta\}$ and $\overline{V_2} = \{\alpha, \gamma\}$, algorithm 3 we would actually find all cubes up to dimension 2.

Correctness. Using the ANF of the function f for the sake of the argument, we see that restricting f to some subset $\overline{V}$ of IV variables will destroy all cubes that contain at least one variable *outside* of this set. On the other hand, all cubes that are fully inside the set $\overline{V}$ will still be found. As this change does not affect the key variables, all superpolys are untouched. Hence, they will be correctly identified as linear, quadratic, ... Using lemma 2 we see that we will only find a fraction of all possible cubes, namely $(|\overline{V}|/|V|)^r$ for a given cube dimension r.

Running time. Giving an exact formula for the running time of the algorithm presented in this section is difficult. However, we can use the same argument as for lemma 2 to see that for any given monomial $\nu \in f$ we only have a probability of $(|\overline{V}|/|V|)^{|\nu|}$ that this monomial is also present in $\overline{f}$. As in Section 5.2, the running time depends on the length of the largest monomial $|\mu|$ and also the number of monomials in the ANF of the function f, denoted by $|f|$. More specifically, we have a running time of $\mathcal{O}(|\mu||f|)$. As both factors are reduced by at least $|\overline{V}|/|V|$, we can expect a reduction in time and memory in the order of $\mathcal{O}((|\overline{V}|/|V|)^2)$. Without having more information about the actual monomials in f, it is not possible to obtain a more accurate bound. To make things worse, the polynomials we are dealing with are also structured. In particular this means that specific subsets of variables can occur much more frequent in the monomials of f than others. So for some choices of $\overline{V}$, our reduction might be much less than expected.

In the case of Trivium, we have replaced the initial function f in 160 variables and (potentially) up to 2^{160} monomials by a function $\overline{f}$ with 80+16=96 key and IV variables. Note that this algorithm is still fully deterministic but cannot find all possible cubes anymore. As we will see in section 5.5 this is only feasible for small cube dimensions. However, we introduce a cube finding algorithm that can cover a higher number of rounds in the next section.

5.4 Going probable

Up to now, we have deterministic algorithms that allows us to find cubes for a given function f or $\overline{f}$, respectively. Still, we can increase the number of rounds we cover by further reducing the number of monomials. As we have dealt with the IV variables in the previous section, we will now concentrate on the key variables. We denote the result by $\overline{\overline{f}}(\overline{K}, \overline{V})$ or simply $\overline{\overline{f}}$. In the case of Trivium, we have a total of 80 key variables, so we can expect a noticeable gain by reducing them.

Unfortunately, the situation is more complicated than in the previous section. By removing key variables, we are actually tempering with the superpoly and therefore destroying relevant information such as its degree. Still, we *can* construct a tester that will output (with some small probability ρ) that a given set of IV variabes $\nu \in \overline{V}$ in f is *not* a cube by looking at the corresponding structure in $\overline{\overline{f}}$. By repeatedly applying this tester on different double-restricted functions $\overline{\overline{f}}_1, \overline{\overline{f}}_2, \ldots$ we can turn this into a tester that detects non-cubes with overwhelming probability.

For simplicity, we only deal with linear fixed cubes here. But both constant and higher order fixed cubes can easily be tested using the very same idea. Before making the algorithm explicit, we discuss what happens to the original function $f(K, V)$ and its restricted version $\overline{f}(K, \overline{V})$ when also restricting the set of key variables K to $\overline{K}$.

Algorithm 4 For a given function f, compute a list of plausible cubes. Parameters are the sub-set of cubes $\overline{V}$, the number of key variables in the test sets κ, and the number of repetitions r.

```
1: function doubleF(V̄, K̄): return f̄̄(V̄, K̄)
2:
3: function plausibleCubes(f, V̄, κ, r)
4: noCubes ← ∅; maybeCubes ← ∅
5: for i ← 1 to r do
6:    K̄ ∈_R {s ⊂ K : |s| = κ} {Get a random subset of size κ} f̄̄ ← doubleF(V̄, K̄)
7:    superPolys ← (0, ..., 0); potentialCubes ← ∅
8:    for all μ ∈ f do
9:       ν, κ ← split(μ)
10:      superPolys_ν ← superPolys_ν + κ
11:      potentialCubes.insert(ν)
12:   end for
13:   noCubes ← noCubes ∪ {ν ∈ potentialCubes : superPolys_ν.degree() > 1}
14:   maybeCubes ← (maybeCubes ∪ potentialCubes) \ noCubes
15: end for
16: return maybeCubes
```

Let $p(K)x_C$ be some superpoly and cube in function f. Assuming $x_C \subset \overline{V}$, the superpoly $p(K)$ becomes $\overline{\overline{p}}(\overline{K})$. Consequently, some monomials in $p(K, \overline{V})$ may be missing in its double-restricted version. On the other hand, $\overline{\overline{p}}(\overline{K})$ being of degree 2 or higher is a *witness* for $p(K)$ being of degree 2 or higher. To see the limits of our algorithm, we consider $p(K)$ be monic and having degree $|K|+1$. No choice of $\overline{K}$ would reveal such a $p(K)$ in $\overline{\overline{p}}(\overline{K})$. Still—it is the same kind of error as above, so if $\overline{\overline{p}}(\overline{K})$ has degree 2 or higher, $p(K)$ is not a superpoly and x_C is certainly not a cube. Still, our algorithm will not detect it. On the other hand, the likelyhood that $p(K)$ has such a structure is very low, so we can detect this case via the degree testing algorithm from section 5.1.

Example 12. Using the same function f as in example 10 and the two sets $\overline{K}_1 := \{a, b\}, \overline{V} := \{\alpha, \beta\}$ of key and IV variables, respectively, we obtain the following function $\overline{\overline{f}}$ in ANF:

$$\overline{\overline{f}}(\overline{K}_1, \overline{V}) = ab\alpha + a\alpha\beta + a\beta + b + a + 1$$

Using algorithm 4, we have the following superpolys, indexed by IV sets:

IV set ν	superPolys$_\nu$	linear
$\emptyset$	$a + b + 1$	$\star$
α	ab	-
β	a	$\star$
$\alpha\beta$	a	$\star$

We see that all but one set is identified as possible cubes, namely $R_1 = \{\emptyset, \beta, \alpha\beta\}$. However, consider the following set of key variables $\overline{K}_2 := \{b, c\}$ we obtain the new function

$$\overline{\overline{f}}(\overline{K}_2, \overline{V}) = b\beta\gamma + bc\alpha + bc\beta + bc + a + b + \beta\gamma + 1$$

and also the following superpolys

IV set ν	superPolys$_\nu$	linear
$\emptyset$	$bc+1$	-
α	bc	-
β	0	$\star$
$\alpha\beta$	0	$\star$

This leads to the new result $R_2 = \{\beta, \alpha\beta\}$ and consequently $R = R_1 \cap R_2 = \{\beta, \alpha\beta\}$. In order to obtain the correct output $\{\beta\}$, we would need to consider the monomial abc, too. Assuming that we cannot develop an ANF with three key variables, we will need to leave this to the degree testing algorithm 2. Hence, we can use algorithm 4 as a filter to identify potentially interesting regions of the overall function f.

Correctness. To see the correctness of algorithm 4, we inspect the ANF of a given function f. For each choice of IV variables $\overline{V}$ and key variables $\overline{K}$, we implicitly set all other variables to zero. Consider the sets $\tilde{K} := K \backslash \overline{K}$, $\tilde{V} := V \backslash \overline{V}$ and the partial assignment $\tau : \tilde{K} \cup \tilde{V} \to 0$ that sets these variables to zero. So we deal with the function $\overline{\overline{f}} := f(\tau, \overline{K}, \overline{V})$. All monomials that contain at least one variable from τ become zero. Consequently, our algorithm will not see them. On the other hand, superpolys that can be expressed solely with variables from the set $\overline{K}$ over cubes from the set $\overline{V}$ will still be found. Hence, our algorithm is a one-sided tester. The probability can be computed using the same ideas as in lemma 2.

Running-Time. As in the previous section, it is difficult to give a closed formula for the running time of the algorithm. However, we can give two sensible bounds on the probability that our algorithm actually outputs a witness. We first bound this probability from below in (2) and then from above (3).

First assume that only *one* coefficient of the superpoly is quadratic, all others are linear. In this case, we need to cover this monomial with our r sets and obtain

$$\frac{|\cup_{i=1}^{r} \{ab : a, b \in K_i\}|}{\binom{|K|}{2}} \tag{2}$$

as lower bound. In theory, we could also assume that $p(K)$ has no quadratic but only one cubic monomial. In practice, this is very unlikely and was not confirmed in our simulations. Still, from a purely theoretical point of view, a probability of zero may be justified here: In this case, $p(K)$ has degree $|K_i| + 1$ but no monomials of degree $2, \ldots, |K_i|$.

On the other hand, we will very likely have a more friendly shape of $p(K)$. Here we assume that for all monomials in K_i that it being present in $p(K)$ with probability 50%. This corresponds to the case where f is well developed in these monomials. This is particularly true for small cube dimensions. Hence we obtain as an upper bound

$$1 - \left(1 - \frac{|K| + 1}{2^{|K|}}\right)^r \tag{3}$$

that a non-linear superpoly is caught by our test. In either case, our practical probability ρ is bounded by both equations and we obtain

$$(2) \leq \rho \leq (3)\,.$$

Note that ρ is a function both of the cube dimension $|C|$ as well as for the current round. So ρ will increase from round to round until we have $\rho = (3)$. On the other hand, increasing the cube dimension $|C|$ will lead to $\rho = (2)$. Unfortunately, the correct value depends on the overall structure of f. Hence, the only realistic way to compute the probability ρ in practice is empirically. All in all, repeating the above test around 80 times gave us a success probability of about ranging from 99% to 30% that C was actually a cube. As expected, the probability was higher for lower rounds than for higher rounds. This was verified with the standard cube test from Section 5.1. More importantly, this cube test has an intrinsic running time of $2^{|C|}$, so precomputing plausible candidates $C_1, \ldots$ is a good way to find all cubes for a given set of IV variables V and cutting down the running time of the combined algorithm.

5.5 Practical implementation

All algorithms from this section were implemented as a hybrid programme in both the computer algebra system Sage [S+13] and the programming language C++; the latter contains the more time-critical parts such as the cube linearity test. In total, the programm consists of 1500 lines of Sage code and the C++ part of around 900 lines of C++ code; both counts include comments.

Using the algorithm 4 with parameters $|\overline{V}| = 15$, $|\overline{K}| = 6$ and 80 iterations per cube, we searched for cubes up to round 570 in Trivium. For example, in round 550 this took around one hour (AMD Opteron 6276@2.3GHz).

Verifying all plausible cubes took up to one additional core-month per round. Consequently, the work was distributed to a cluster with 256 cores and 1TB of RAM. Interestingly, with the above parameters, we could go on forever as the total number of variables per iteration is only 21. Alas, from round 570 on, there are no cubes left with 15 or less IV variables. On the other hand, reducing the number of key bits from 6 to 5 (or even lower) gave us too many false positives that the algorithm became unusable in practice. Unfortunately, the implementation consumed a lot of disk space (up to 1.2TB), so we could not increase the number of variables or iterations further.

In contrast, using the exact algorithm 3, we were able to *fully* cover the first 446 rounds of Trivium. To this aim, we needed to process a Boolean function with around 50 million monomials. To put this into perspective: A single state bit in round 399 needs around 2.7 GB of disk space and consists of about 82 million monomials with a total of 675 million appearances of the 160 variables from $K \cup V$. Unfortunately, Sage and in particular its polynomial package PolyBoRi [BD09] are not able to deal with these high numbers of monomials / variables, so PolyBoRi repeatedly and predictably crashed. To conclude: memory was the real bottle neck for this algorithm, not time. The most promising way seems a full rewrite of the algorithm in C++. This also includes the development of a more compact representation of the monomials.

Download. Raw data for fixed and free cubes up to round 446 are are available for download at `www.cits.rub.de/imperia/md/content/wolf/cubelistfixed.zip` and `www.cits.rub.de/imperia/md/content/wolf/cubelistfree.zip`

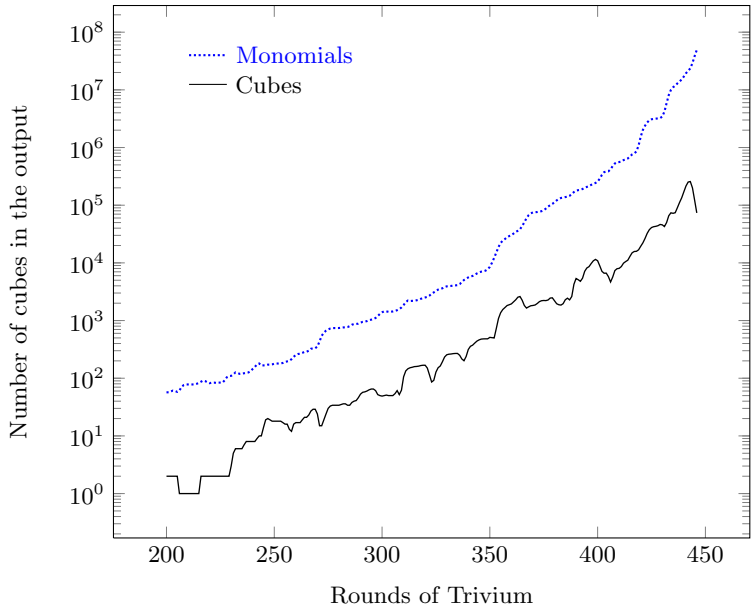


Fig. 4. Both the number of cubes and also the total number of monomials in the output function $f(V, K)$ of Trivium, plotted against the number of rounds.

6 Practical Results on Cubes

We now use the algorithms from the previous section to get some insights on the distribution of cubes for Trivium. This also serves as a test to see if both definitions and algorithms are actually useful in practice. All results were obtained on the computing cluster described in Section 5.5.

6.1 Cubes in the output

In figure 4 (cf. also table 2) we see the total number of cubes in the output function

$$z_i = c_{i-65} + c_{i-110} + a_{i-65} + a_{i-92} + b_{i-68} + b_{i-83}$$

of Trivium for $200 \leq i < 447$. Note that values below 200 are of only limited interest and hence not plotted here. This figure was computed with algorithm 3. Therefore, we know that we have the *exact* number of cubes here. Moreover, we have verified for randomly selected cubes that they actually have the same degree as the superpoly that was output by the algorithm (BLR test). For comparison, we have also plotted in figure 4 the number of monomials. These monomials are in the variables from $V \cup K$ for $V = \{v_0, \ldots, v_{79}\}$ and $K = \{k_0, \ldots, k_{79}\}$. In addition, we can see both the degree of the output function and also the average degree of all monomials in figure 5.

There are two interesting features here: First, the number of cubes increases exponentially. Secondly, the plot of the number of cubes per round is much more noisy than the corresponding plot for the number of monomials. All in all, it is very hard to predict the number of cubes for a given round. In particular, it is not clear from the above curve if the number of cubes will increase exponentially up to round 1152 or if it will decrease at some state. The latter is justified as the

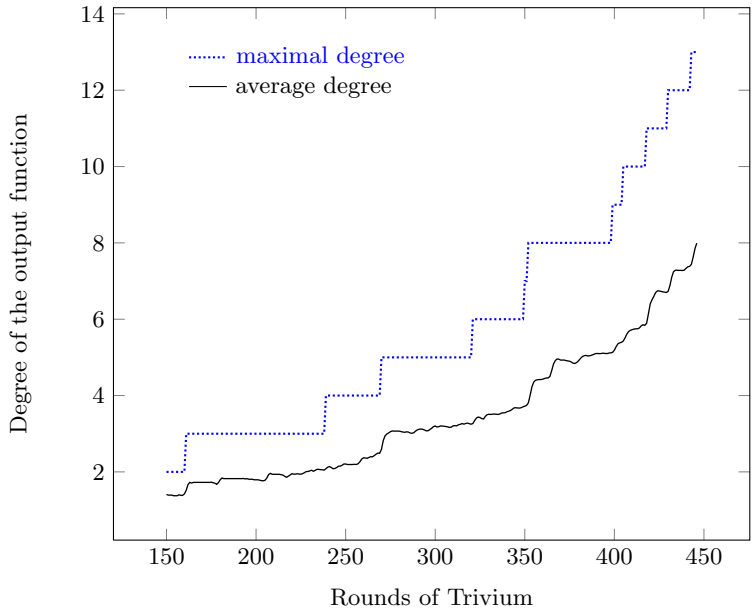


Fig. 5. Degree of the output function of Trivium.

number of possible monomials is bounded from above by $2^{|K|+|V|}$. In the case of Trivium, we have 2^{160}. To this aim, we have a closer look at the actual dimension of cubes, cf. figure 6. Here, we see the same cubes as in figure 3 but do distinguish them by their dimension. To keep figure 6 readable, we did restrict to dimensions $1, 2, 6, 10$ and 11. A list of *all* cubes by dimension is given in table 3. First we note that some cube dimensions only occur after a particular round. For example, we do not have cubes of dimension 10 before round 430 (dimension 11: round 439). This finding is in line with the overall definition of Trivium: All state bits are initialized with polynomials of degree -1, 0 or 1. The output function of Trivium does not change this as it only contains addition, no multiplication. However, the update function of Trivium for each inidividual state contains *exactly* one multiplication of two previously computed state bits. Hence, the overall degree of the corresponding polynomial will most likely increase here and give rise to new (potential) cubes. In addition we notice that we do not have any cubes of dimension 1 after round 394 and that there are rounds without cubes of dimension 1 from round 362 onwards. To tackle this question, we have a closer look how cubes develop in the following section.

Regarding the number of monomials we see the corresponding plot is far smoother and we have at least an order of magnitude more monomials than cubes for a given round. Even though, giving a closed formula or even extrapolating is difficult enough.

6.2 Life-cycle of a Cube

For the sake of the argument, we restrict to fixed cubes that have a superpoly of *exactly* degree 1. Keep in mind that every cube corresponds to monomials that share the same variables from the set V. So when viewing a cube from round to round (as in figure 4), we can identify three phases in the life of such a cube—each corresponding to the existence and non-existence of certain monomials in the ANF of f:

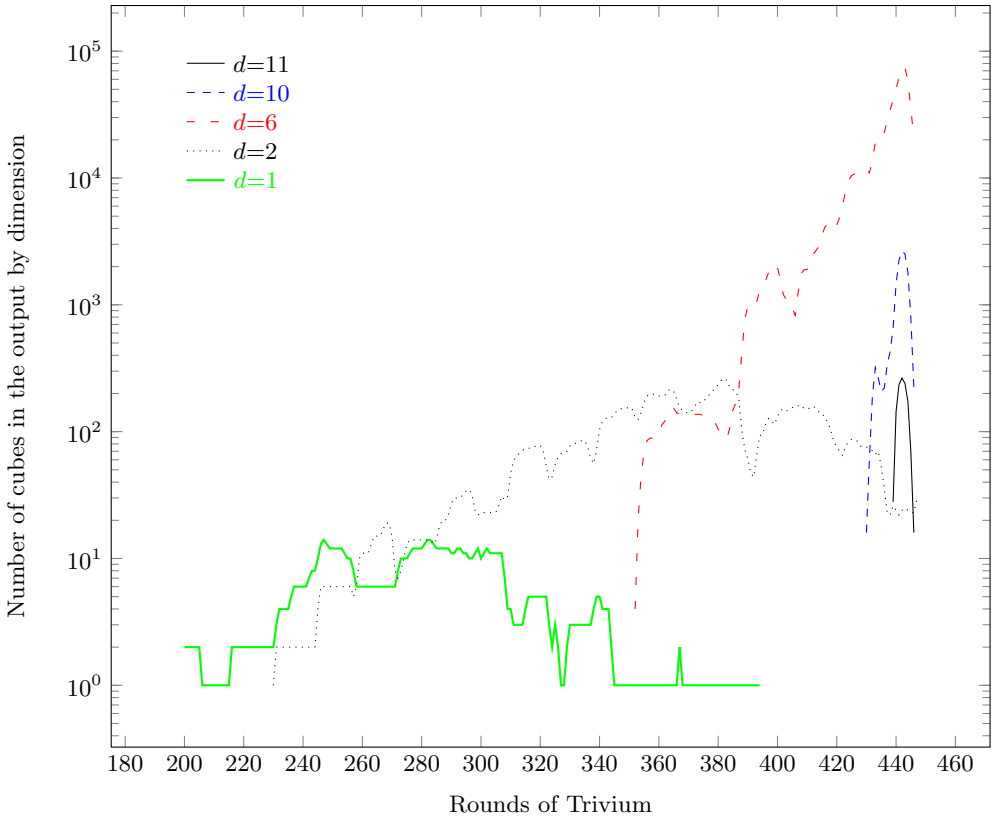


Fig. 6. Number of cubes of dimension d in the output of Trivium, plotted per dimension $d = 1, 2, 6, 10, 11$ against the number of rounds.

1. Non-existence
2. Creation
3. Destruction

In phase 1, a cube C cannot exist as there is no monomial that contains the corresponding variables from the set of IV variables V, plus exactly one variable from the set of key variables K. As soon as this changes, we are in phase 2. Now, we have at least one monomial that is of the form $x_C k_i$ with $k_i \in K$ and $C \subset V$. Finally, we may have a monomial of the form $x_C k_i k_j$ with $0 \leq i, j < 80, i \neq j$. This will destroy the cube, and we are in phase 3. Note that any monomial with at least two key variables will do to destroy the corresponding cube x_C. Alternatively, we can use the definitions of section 2.3 and say that a cube is constant (one or negative) is phase 1, linear in phase 2, and a higher-order (quadratic, cubic) cube in phase 3. We illustrate this with the following example.

Example 13. Let $K := \{a, b, c\}, V := \{\alpha, \beta\}$ be the sets of key and IV variables, respectively. Moreover, consider the following function $f : K \cup V \to \mathbb{B}$ in algebraic normal form (ANF):

$$f(K, V) := a\alpha + c\alpha + c\alpha\beta + bc\alpha\beta + a$$

Using the above definition, we have the following superpolys, indexed by IV sets:

IV set	superPolys	degree
$\emptyset$	a	1
α	$a+c$	1
β	0	-1
$\alpha\beta$	bc	2

For α, we have a superpoly of exactly degree 1. Hence, this is a cube in phase 2. In contrast, we do not have a monomial for the potential cube β yet. Hence, this cube is in phase 1. Finally, the superpoly associated to $\alpha\beta$ is of degree 2. Consequently, this is a cube in phase 3.
Now, multiplying f with the function $g(K,V) = b\alpha + \beta$ yields

$$fg = a\alpha\beta + a\beta + bc\alpha + bc\alpha\beta\,.$$

This leads to the corresponding cubes in $\alpha, \beta, \alpha\beta$, respectively:

IV set	superPolys	degree
α	bc	2
β	a	1
$\alpha\beta$	$bc+a$	2

So the cube α goes from phase 2 to phase 1, the cube β from phase 1 to phase 2 and $\alpha\beta$ stays in phase 3.

Neither from the above example nor from the above definition it is evident that all cubes will go through the cycle $1 \to 2 \to 3$ without jumping back. In particular, it were possible that (w.l.o.g.) the two state bits c_{i-110}, a_{i-65} both contain the same monomial $x_C k_i$ for some $C \subset V$ and $0 \leq i < 80$. By the definition of the output function, they would cancel out so the cube C would go back from phase 2 to phase 1.

In practice, we encountered such a behaviour in our simulations only very rarely. Moreover, the very structure of the Trivium update and output functions makes it very difficult to actually construct such examples.

Conjecture 1. Let $C \subset V$ be a set of IV variables for Trivium. Either, C never forms a cube within Trivium. Alternatively there exists a unique pair $R_c, R_d \in \mathbb{N}$ with $R_c < R_d$ and R_c being the maximal and R_d being the minimal number such that for $R \in \mathbb{N}$ with $R < R_c$ or $R > R_d$ we do not have a cube for the set C in the output of the corresponding round of Trivium. We call the rounds r with $R_c \leq r \leq R_d$ are called the *living room* of the cube C.

This conjecture is a direct consequence of the life-cycle described above. It also includes the (rare) case that a cube directly jumps from phase 1 to phase 3.

6.3 State cubes

As already mentioned in Section 2.5, we conjecture the existence of state cubes. As we see in figure 7, this is actually correct. Moreover, the total number of state cubes (register A+B+C) is always higher than the number of cubes in the output for a given round R. Note that figure 7 plots all values from round 150 up to the maximal number we could compute. As the polynomials associated to state bits are larger than the ones associated to output bits, we only manage 397 rounds for state bits but up to 446 rounds for output bits.

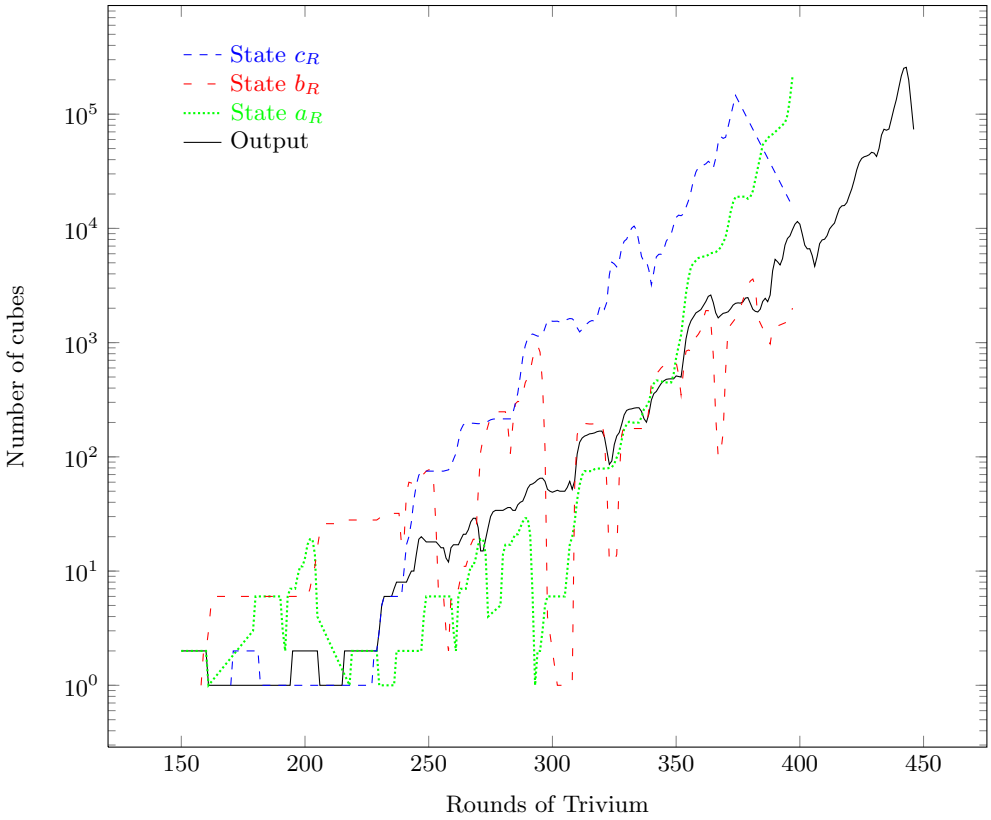


Fig. 7. Total number of cubes both in the highest state bit a_R, b_R, c_R and the output, plotted by round R.

We want to note an interesting connection between output and state cubes. This connection is only true for ciphers that use a linear output function, *e.g.* Trivium. More specifically, each cube in the output must first exist as a cube in the state: As the output function is fully linear, it cannot create new cubes but only collect cubes found so far in the corresponding state bits. The only change such a linear output function can do is from phase 2 to phase 3: If one state contains a linear cube x_C and another contains a monomial x_C but with a superpoly of higher degree, the cube x_C will vanish in the output.

In addition, the output function depends on state bits from round $R-66$ or even earlier, so the number of cubes in the output will simply follow the number of cubes in the state by some delay factor. As the output function is constructed deliberately irregular, it is not easy to exploit this delay in practice. Still—in the case of Trivium, finding state cubes helps to find output cube. Alas, it is difficult to model the cube cancellation mentioned in the previous section, so predicting the number of cubes for a given round is left as on open problem.

7 Conclusions

The contributions of this article are manyfold. Firstly, we have dealt with a thorough analysis of the different variants of cubes in Section 2. In summary, we have found several interesting dimensions and put them into one framework:

- The first classification uses the **degree** of the superpoly. In particular, *linear* cubes are useful for cryptanalytic purposes. In some cases, *quadratic* and maybe *cubic* cubes are, too. Specifically for state cubes, *constant* cubes such as *negative* or *one* cubes can help to speed up a cryptanalytic attack.
- By **fixing** some IV variables, we have a full continuum of possible cubes, ranging from *free* cubes via *flexible* cubes to *fixed*, and *zero* cubes, respectively.
- Taking the internal working of the cipher into account, we have **state** cubes as internal invariants of several instances.
- Reducing the overall degree of the superpoly is the aim of **dynamic** cubes. They were successful employed against Grain-128.
- A second way to achieve this aim are **mixed** cubes. Here, we sum over several cubes of higher key-degree to achieve cubes of lower key-degree.
- Taking mixed cubes one step further, we obtain **factor** cubes. Interestingly, they cannot be distinguished from ordinary cubes when using black-box methods such as degree testing (Section 5.1).

All these dimensions can be grouped freely, *e.g.* we can have flexible dynamic cubes as well as linear free factor state cubes which helps us to develop new and more powerful cube attacks. To this end, we have developed an **algebraic modelling technique** in Section 4. Using both *state cubes* and *similar variables*, we considerably reduced the amount of intermediate variables needed in the modelling of Trivium. In particular, we have empirically found a *saturation* both for variables and monomials. The corresponding algorithm is generic and can also be applied to other ciphers. We are confident that they will also show a saturation effect, although further research is needed to confirm this. In addition, our algebraic modelling greatly benefits from the existence of *state cubes* in Trivium. We have seen in section 6.3 and in particular figure 7 that plenty exist in Trivium.

Various **algorithms** to find cubes and identify their degree are discussed in section 5. The *degree identification algorithm* uses black-box access to a given Boolean function f. In contrast, the other algorithms from this section use access to the algebraic normal form of a full or restricted version of this Boolean function f. In contrast to previously known algorithms such as *random walk*, this allows a precise calculation of the number of cubes up to a certain round. Depending on the algorithm we use, our implementation can handle up to 446 or 570 rounds, respectively.

These algorithms were then used to explore **specific properties of Trivium** in Section 6. In particular, we have presented for the first time the exact **number of cubes** up to round 446, cf. figure 4 and table 2. This will hopefully better our understanding of both the strengths and the limitations of cube attacks. To this aim, we have also given a *detailed list of cubes per dimension* in figure 6 and table 3, respectively. Based on these findings, we have developed a theory of the **life-cycle** of cubes and made the connection between state cubes and output cubes explicit.

All in all, the aim of this article was to present a coherent theory of cubes, show that it has applications in practice and shine light on the structure of Trivium. While this was successful to some extent, there are still **open questions** that need to be addressed. Firstly, we cannot efficiently test of some of the cube properties such as polycubes or factor cubes. Here, a deeper understanding

of the corresponding algorithms are needed. Secondly, finding a full list of cubes up to a given dimension is currently seriously limited by our implementation. Doing a full implementation could raise the bar here. Thirdly, the algebraic modelling could be aided by a specific equation solving algorithm. Finally, predicting the number of cubes (and also their dimension) for higher rounds would be beneficial.

Acknowledgements

The first author wants to thank Wolfram Koepf (Kassel) for fruitful discussions and guidance. The second author gratefully acknowledges an Emmy Noether Grant of the Deutsche Forschungsgemeinschaft (DFG). In addition, the authors want to thank Saqib Kakvi (Bochum) for helpful remarks and Marina Efimenko (Bochum) for help with table 3.

References

[ADMS09] Jean-Philippe Aumasson, Itai Dinur, Willi Meier, and Adi Shamir. Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium. In *FSE*, pages 1–22, 2009.

[BD09] Michael Brickenstein and Alexander Dreyer. PolyBoRi: A framework for Groebner-basis computations with Boolean polynomials. *Journal of Symbolic Computation*, 44(9):1326 – 1345, 2009. Effective Methods in Algebraic Geometry.

[BLR93] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-Testing/Correcting with Applications to Numerical Problems. *J. Comput. Syst. Sci.*, 47(3):549–595, 1993.

[CMR05] Carlos Cid, Sean Murphy, and Matthew J. B. Robshaw. Small scale variants of the aes. In *FSE*, volume 3557 of *Lecture Notes in Computer Science*, pages 145–162. Henri Gilbert and Helena Handschuh, editors, Springer, 2005. ISBN 3-540-26541-4.

[CP08] C. De Cannire and B. Prenel. Trivium. In *New Stream Cipher Designs*, volume 4986 of *LNCS*, pages 84–97. Springer, 2008.

[DAS11] Itai Dinur and Adi Adi Shamir. Breaking Grain-128 with Dynamic Cube Attacks. In *18th International Workshop Fast Software Encryption, FSE 2011*, volume 6733, pages 167–187. Springer, 2011.

[DS09a] I. Dinur and A. Shamir. Cube Attacks on tweakable black box polynomials. In *EUROCRYPT. Lecture Notes in Computer Science*, volume 5479, pages 278–299. Springer, 2009.

[DS09b] Itai Dinur and Adi Shamir. Side Channel Cube Attacks on Block Ciphers. Cryptology ePrint Archive, Report 2009/127, 2009. http://eprint.iacr.org/2009/127/, 15 pages.

[FV13] P.A. Fouque and T. Vannet. Improving Key Recovery to 784 and 799 rounds of Trivium using Optimized Cube Attacks. FSE 2013, Fast Software Encryption, pp, 2013.

[HJMM06] M. Hell, T. Johannson, A. Maximov, and W. Meier. A stream cipher proposal: Grain-128. In *IEEE International Symposium on Information Theory (ISIT 2006)*, 2006.

[JPRZ09] Charanjit S. Jutla, Anindya C. Patthak, Atri Rudra, and David Zuckerman. Testing low-degree polynomials over prime fields. In *Random Struct. Algorithms*, pages 163–193, 2009. http://www.cs.utexas.edu/~diz/pubs/low_deg_test.pdf.

[KHK06] Shahram Khazaei, Mahdi M. Hasanzadeh, and Mohammad S. Kiaei. Linear Sequential Circuit Approximation of Grain and Trivium Stream Ciphers. Cryptology ePrint Archive, Report 2006/141, 2006. http://eprint.iacr.org/2006/141/.

[KMNP11] Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional Differential Cryptanalysis of Trivium and KATAN. In *Selected Areas in Cryptography*, pages 200–212, 2011.

[Lai94] X. Lai. Higher order derivatives and differential cryptanalysis. In *Symposium on Communication, Coding and Cryptography in honor of James L. Massey on the occasion of his 60th birthday*, pages 227–233, 1994.

[Lat09] Joel Lathrop. Cube Attack on Cryptographic Hash Functions, 2009. Master thesis; Rochester Institute of Technology.

[Lea10] Gregor Leander. Small scale variants of the block cipher Present. Cryptology ePrint Archive, Report 2010/143, 2010.

[MR02] Sean Murphy and Matthew J.B. Robshaw. Essential algebraic structure within the AES. In *Advances in Cryptology — CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 1–16. Moti Yung, editor, Springer, 2002.

[MS09] Piotr Mroczkowski and Janusz Szmidt. Cube Attack on Courtois Toy Cipher. Cryptology ePrint Archive, Report 2009/497, 2009. http://eprint.iacr.org/2009/497/.

[MS11] Piotr Mroczkowski and Janusz Szmidt. Corrigendum to: The Cube Attack on Stream Cipher Trivium and Quadraticity Tests. Cryptology ePrint Archive, Report 2011/032, 2011. http://eprint.iacr.org/2011/032/.

[Rad06] H. Raddum. Cryptanalytic results on Trivium. http://www.ecrypt.eu.org/stream/triviump3.html, 2006.

[S+13] W. A. Stein et al. *Sage Mathematics Software (Version 5.7)*. The Sage Development Team, 2013. http://www.sagemath.org.

[SFP08] Ilaria Simonetti, Jean-Charles Faugre, and Ludovic Perret. Algebraic Attack Against Trivium. In *First International Conference on Symbolic Computation and Cryptography, SCC 08*, LMIB, pages 95–102, Beijing, China, April 2008. http://www-polsys.lip6.fr/~jcf/Papers/SCC08c.pdf.

[SR12] T.E. Schilling and H. Raddum. Analysis of Trivium using compressed right hand side equations. In *Information Security and Cryptology. Lecture Notes in Computer Science*, volume 7259, pages 18–32. Springer, 2012.

[Sta10] Paul Stankovski. Greedy Distinguishers and Nonrandomness Detectors. In *INDOCRYPT*, pages 210–226, 2010.

[T+13] S. Teo et al. Algebraic analysis of Trivium-like ciphers, 2013. http://www.eprint.iacr.org/2013/240.pdf.

[Vie07] Michael Vielhaber. Breaking ONE.FIVIUM by AIDA an Algebraic IV Differential Attack. Cryptology ePrint Archive, Report 2007/413, 2007. http://eprint.iacr.org/2007/413/.

[Vie08] Michael Vielhaber. TRIVIUM's output partially autocancels. Cryptology ePrint Archive, Report 2008/377, 2008. http://eprint.iacr.org/2008/377/.

[YLWQ09] Lin Yang, Lin, Meiqin Wang, and Siyuan Qiao. Side Channel Cube Attack on PRESENT. In *Proceedings of the 8th International Conference on Cryptology and Network Security*, CANS '09, pages 379–391, Berlin, Heidelberg, 2009. Springer-Verlag.

A Full list of output cubes

Table 2. Number of all cubes (#) in the output of Trivium from round R=200 to 446.

R	#	R	#	R	#	R	#	R	#	R	#	R	#
200:	2	201:	2	202:	2	203:	2	204:	2	205:	2	206:	1
215:	1	216:	2	217:	2	218:	2	219:	2	220:	2	221:	2
222:	2	223:	2	224:	2	225:	2	226:	2	227:	2	228:	2
229:	2	230:	3	231:	5	232:	6	233:	6	234:	6	235:	6
236:	7	237:	8	238:	8	239:	8	240:	8	241:	8	242:	9
243:	10	244:	10	245:	14	246:	19	247:	20	248:	19	249:	18
250:	18	251:	18	252:	18	253:	18	254:	17	255:	16	256:	16
257:	13	258:	12	259:	16	260:	17	261:	17	262:	17	263:	19
264:	21	265:	21	266:	23	267:	27	268:	29	269:	29	270:	24
271:	15	272:	15	273:	19	274:	24	275:	30	276:	33	277:	34
278:	34	279:	34	280:	34	281:	35	282:	36	283:	36	284:	34
285:	34	286:	38	287:	40	288:	41	289:	46	290:	53	291:	57
292:	58	293:	60	294:	63	295:	65	296:	65	297:	61	298:	52
299:	50	300:	49	301:	50	302:	51	303:	50	304:	50	305:	50
306:	54	307:	61	308:	52	309:	62	310:	106	311:	134	312:	146
313:	152	314:	155	315:	159	316:	160	317:	162	318:	166	319:	168
320:	168	321:	150	322:	114	323:	86	324:	92	325:	128	326:	152
327:	163	328:	193	329:	233	330:	255	331:	261	332:	263	333:	267
334:	269	335:	269	336:	250	337:	216	338:	201	339:	238	340:	315
341:	358	342:	374	343:	407	344:	440	345:	462	346:	477	347:	481
348:	482	349:	482	350:	511	351:	505	352:	499	353:	729	354:	1,083
355:	1,364	356:	1,552	357:	1,680	358:	1,821	359:	1,880	360:	1,946	361:	2,110
362:	2,290	363:	2,542	364:	2,611	365:	2,248	366:	1,824	367:	1,646	368:	1,729
369:	1,805	370:	1,826	371:	1,862	372:	1,976	373:	2,129	374:	2,208	375:	2,229
376:	2,217	377:	2,272	378:	2,454	379:	2,479	380:	2,202	381:	1,958	382:	1,880
383:	1,855	384:	1,962	385:	2,294	386:	2,442	387:	2,283	388:	2,609	389:	4,252
390:	5,361	391:	5,101	392:	4,788	393:	5,426	394:	7,140	395:	8,160	396:	8,604
397:	9,727	398:	10,855	399:	11,488	400:	10,891	401:	8,879	402:	7,130	403:	6,618
404:	6,643	405:	5,778	406:	4,662	407:	5,682	408:	7,358	409:	7,924	410:	8,039
411:	8,634	412:	9,878	413:	10,531	414:	11,126	415:	12,978	416:	14,946	417:	15,760
418:	15,859	419:	16,847	420:	19,430	421:	22,250	422:	26,575	423:	32,536	424:	37,681
425:	40,997	426:	42,391	427:	43,025	428:	44,104	429:	46,352	430:	45,641	431:	42,639
432:	50,108	433:	65,663	434:	73,800	435:	72,340	436:	74,086	437:	90,529	438:	111,375
439:	134,302	440:	170,820	441:	215,812	442:	252,203	443:	257,132	444:	199,296	445:	121,891
446:	73,540												

B Output cubes by dimension

Table 3: Number of cubes per dimension in the output of Trivium from round R=200 to 446. Cells with a dot ('.') indicate no cubes cubes for this round/dimension.

Round	sum	1	2	3	4	5	6	7	8	9	10	11
200	2	2	.	.	.	.	.	.	.	.	.	.
201	2	2	.	.	.	.	.	.	.	.	.	.
202	2	2	.	.	.	.	.	.	.	.	.	.
203	2	2	.	.	.	.	.	.	.	.	.	.
204	2	2	.	.	.	.	.	.	.	.	.	.
205	2	2	.	.	.	.	.	.	.	.	.	.
206	1	1	.	.	.	.	.	.	.	.	.	.
215	1	1	.	.	.	.	.	.	.	.	.	.
216	2	2	.	.	.	.	.	.	.	.	.	.
217	2	2	.	.	.	.	.	.	.	.	.	.
218	2	2	.	.	.	.	.	.	.	.	.	.
219	2	2	.	.	.	.	.	.	.	.	.	.
220	2	2	.	.	.	.	.	.	.	.	.	.
221	2	2	.	.	.	.	.	.	.	.	.	.
222	2	2	.	.	.	.	.	.	.	.	.	.
223	2	2	.	.	.	.	.	.	.	.	.	.
224	2	2	.	.	.	.	.	.	.	.	.	.
225	2	2	.	.	.	.	.	.	.	.	.	.
226	2	2	.	.	.	.	.	.	.	.	.	.
227	2	2	.	.	.	.	.	.	.	.	.	.
228	2	2	.	.	.	.	.	.	.	.	.	.
229	2	2	.	.	.	.	.	.	.	.	.	.
230	3	2	1	.	.	.	.	.	.	.	.	.
231	5	3	2	.	.	.	.	.	.	.	.	.
232	6	4	2	.	.	.	.	.	.	.	.	.
233	6	4	2	.	.	.	.	.	.	.	.	.
234	6	4	2	.	.	.	.	.	.	.	.	.
235	6	4	2	.	.	.	.	.	.	.	.	.
236	7	5	2	.	.	.	.	.	.	.	.	.
237	8	6	2	.	.	.	.	.	.	.	.	.
238	8	6	2	.	.	.	.	.	.	.	.	.
239	8	6	2	.	.	.	.	.	.	.	.	.
240	8	6	2	.	.	.	.	.	.	.	.	.
241	8	6	2	.	.	.	.	.	.	.	.	.
242	9	7	2	.	.	.	.	.	.	.	.	.
243	10	8	2	.	.	.	.	.	.	.	.	.
244	10	8	2	.	.	.	.	.	.	.	.	.
245	14	10	4	.	.	.	.	.	.	.	.	.
246	19	13	6	.	.	.	.	.	.	.	.	.
247	20	14	6	.	.	.	.	.	.	.	.	.
248	19	13	6	.	.	.	.	.	.	.	.	.
249	18	12	6	.	.	.	.	.	.	.	.	.
250	18	12	6	.	.	.	.	.	.	.	.	.
251	18	12	6	.	.	.	.	.	.	.	.	.
252	18	12	6	.	.	.	.	.	.	.	.	.
253	18	12	6	.	.	.	.	.	.	.	.	.

Continued on next page

Table 3 – continued from previous page

Round	sum	1	2	3	4	5	6	7	8	9	10	11
254	17	11	6	.	.	.	.	.	.	.	.	.
255	16	10	6	.	.	.	.	.	.	.	.	.
256	16	10	6	.	.	.	.	.	.	.	.	.
257	13	8	5	.	.	.	.	.	.	.	.	.
258	12	6	6	.	.	.	.	.	.	.	.	.
259	16	6	10	.	.	.	.	.	.	.	.	.
260	17	6	11	.	.	.	.	.	.	.	.	.
261	17	6	11	.	.	.	.	.	.	.	.	.
262	17	6	11	.	.	.	.	.	.	.	.	.
263	19	6	13	.	.	.	.	.	.	.	.	.
264	21	6	15	.	.	.	.	.	.	.	.	.
265	21	6	15	.	.	.	.	.	.	.	.	.
266	23	6	15	2	.	.	.	.	.	.	.	.
267	27	6	17	4	.	.	.	.	.	.	.	.
268	29	6	19	4	.	.	.	.	.	.	.	.
269	29	6	19	4	.	.	.	.	.	.	.	.
270	24	6	15	3	.	.	.	.	.	.	.	.
271	15	6	8	1	.	.	.	.	.	.	.	.
272	15	8	7	.	.	.	.	.	.	.	.	.
273	19	10	8	1	.	.	.	.	.	.	.	.
274	24	10	10	4	.	.	.	.	.	.	.	.
275	30	10	13	7	.	.	.	.	.	.	.	.
276	33	11	14	8	.	.	.	.	.	.	.	.
277	34	12	14	8	.	.	.	.	.	.	.	.
278	34	12	14	8	.	.	.	.	.	.	.	.
279	34	12	14	8	.	.	.	.	.	.	.	.
280	34	12	14	8	.	.	.	.	.	.	.	.
281	35	13	14	8	.	.	.	.	.	.	.	.
282	36	14	14	8	.	.	.	.	.	.	.	.
283	36	14	14	8	.	.	.	.	.	.	.	.
284	34	13	13	8	.	.	.	.	.	.	.	.
285	34	12	14	8	.	.	.	.	.	.	.	.
286	38	12	18	8	.	.	.	.	.	.	.	.
287	40	12	20	8	.	.	.	.	.	.	.	.
288	41	12	20	9	.	.	.	.	.	.	.	.
289	46	12	22	12	.	.	.	.	.	.	.	.
290	53	11	27	15	.	.	.	.	.	.	.	.
291	57	11	30	16	.	.	.	.	.	.	.	.
292	58	12	30	16	.	.	.	.	.	.	.	.
293	60	12	30	18	.	.	.	.	.	.	.	.
294	63	11	32	20	.	.	.	.	.	.	.	.
295	65	11	34	20	.	.	.	.	.	.	.	.
296	65	10	35	20	.	.	.	.	.	.	.	.
297	61	10	32	19	.	.	.	.	.	.	.	.
298	52	11	24	17	.	.	.	.	.	.	.	.
299	50	12	22	16	.	.	.	.	.	.	.	.
300	49	10	23	16	.	.	.	.	.	.	.	.
301	50	11	23	16	.	.	.	.	.	.	.	.
302	51	12	23	16	.	.	.	.	.	.	.	.
303	50	11	23	16	.	.	.	.	.	.	.	.
304	50	11	23	16	.	.	.	.	.	.	.	.
305	50	11	23	16	.	.	.	.	.	.	.	.

Continued on next page

Table 3 – continued from previous page

Round	sum	1	2	3	4	5	6	7	8	9	10	11
306	54	11	26	17	.	.	.	.	.	.	.	.
307	61	11	31	19	.	.	.	.	.	.	.	.
308	52	7	29	16	.	.	.	.	.	.	.	.
309	62	4	31	22	5	.	.	.	.	.	.	.
310	106	4	47	40	15	.	.	.	.	.	.	.
311	134	3	59	51	21	.	.	.	.	.	.	.
312	146	3	64	56	23	.	.	.	.	.	.	.
313	152	3	68	58	23	.	.	.	.	.	.	.
314	155	3	71	58	23	.	.	.	.	.	.	.
315	159	4	74	58	23	.	.	.	.	.	.	.
316	160	5	74	58	23	.	.	.	.	.	.	.
317	162	5	74	60	23	.	.	.	.	.	.	.
318	166	5	76	62	23	.	.	.	.	.	.	.
319	168	5	78	62	23	.	.	.	.	.	.	.
320	168	5	78	62	23	.	.	.	.	.	.	.
321	150	5	70	56	19	.	.	.	.	.	.	.
322	114	5	54	44	11	.	.	.	.	.	.	.
323	86	3	42	34	7	.	.	.	.	.	.	.
324	92	2	44	35	11	.	.	.	.	.	.	.
325	128	3	55	49	21	.	.	.	.	.	.	.
326	152	2	63	60	27	.	.	.	.	.	.	.
327	163	1	67	62	31	2	.	.	.	.	.	.
328	193	1	67	70	47	8	.	.	.	.	.	.
329	233	2	70	82	65	14	.	.	.	.	.	.
330	255	3	77	88	71	16	.	.	.	.	.	.
331	261	3	81	90	71	16	.	.	.	.	.	.
332	263	3	81	92	71	16	.	.	.	.	.	.
333	267	3	83	94	71	16	.	.	.	.	.	.
334	269	3	85	94	71	16	.	.	.	.	.	.
335	269	3	85	94	71	16	.	.	.	.	.	.
336	250	3	76	88	67	16	.	.	.	.	.	.
337	216	3	62	76	59	16	.	.	.	.	.	.
338	201	4	56	70	55	16	.	.	.	.	.	.
339	238	5	72	86	59	16	.	.	.	.	.	.
340	315	5	104	122	68	16	.	.	.	.	.	.
341	358	4	120	144	74	16	.	.	.	.	.	.
342	374	4	127	147	78	18	.	.	.	.	.	.
343	407	4	130	155	94	24	.	.	.	.	.	.
344	440	2	129	167	112	30	.	.	.	.	.	.
345	462	1	137	174	118	32	.	.	.	.	.	.
346	477	1	148	178	118	32	.	.	.	.	.	.
347	481	1	151	179	118	32	.	.	.	.	.	.
348	482	1	152	179	118	32	.	.	.	.	.	.
349	482	1	152	179	118	32	.	.	.	.	.	.
350	511	1	157	192	125	36	.	.	.	.	.	.
351	505	1	150	190	124	40	.	.	.	.	.	.
352	499	1	129	181	132	52	4	.	.	.	.	.
353	729	1	124	253	220	111	20	.	.	.	.	.
354	1,083	1	141	358	346	192	45	.	.	.	.	.
355	1,364	1	171	434	438	251	69	.	.	.	.	.
356	1,552	1	191	484	500	291	85	.	.	.	.	.
357	1,680	1	198	525	551	316	89	.	.	.	.	.

Continued on next page

Table 3 – continued from previous page

Round	sum	1	2	3	4	5	6	7	8	9	10	11
358	1,821	1	198	585	610	338	89	.	.	.	.	.
359	1,880	1	191	613	634	352	89	.	.	.	.	.
360	1,946	1	191	621	664	372	97	.	.	.	.	.
361	2,110	1	193	655	733	415	113	.	.	.	.	.
362	2,290	.	193	711	810	455	121	.	.	.	.	.
363	2,542	.	210	776	904	519	133	.	.	.	.	.
364	2,611	.	217	753	923	565	153	.	.	.	.	.
365	2,248	.	207	582	789	517	153	.	.	.	.	.
366	1,824	1	180	425	633	444	141	.	.	.	.	.
367	1,646	2	154	364	567	422	137	.	.	.	.	.
368	1,729	1	148	407	595	441	137	.	.	.	.	.
369	1,805	.	142	441	624	461	137	.	.	.	.	.
370	1,826	.	140	447	637	465	137	.	.	.	.	.
371	1,862	.	144	459	649	473	137	.	.	.	.	.
372	1,976	.	157	502	689	491	137	.	.	.	.	.
373	2,129	.	172	559	748	513	137	.	.	.	.	.
374	2,208	.	169	595	780	527	137	.	.	.	.	.
375	2,229	.	171	618	796	511	133	.	.	.	.	.
376	2,217	.	186	643	797	466	125	.	.	.	.	.
377	2,272	.	195	685	824	447	121	.	.	.	.	.
378	2,454	.	207	756	897	473	121	.	.	.	.	.
379	2,479	.	221	770	908	463	117	.	.	.	.	.
380	2,202	.	234	663	802	398	105	.	.	.	.	.
381	1,958	.	253	577	698	337	93	.	.	.	.	.
382	1,880	.	264	556	656	315	89	.	.	.	.	.
383	1,855	.	252	551	649	314	89	.	.	.	.	.
384	1,962	.	220	552	697	384	109	.	.	.	.	.
385	2,294	.	214	594	822	515	149	.	.	.	.	.
386	2,442	.	208	610	878	577	169	.	.	.	.	.
387	2,283	.	180	519	768	575	221	20	.	.	.	.
388	2,609	.	101	407	776	818	421	86	.	.	.	.
389	4,252	.	72	570	1,210	1,434	771	195	.	.	.	.
390	5,361	.	61	711	1,518	1,836	977	258	.	.	.	.
391	5,101	.	47	633	1,385	1,781	985	270	.	.	.	.
392	4,788	.	44	577	1,266	1,678	953	270	.	.	.	.
393	5,426	1	65	662	1,463	1,890	1,055	290	.	.	.	.
394	7,140	1	93	937	2,014	2,458	1,307	330	.	.	.	.
395	8,160	.	95	1,114	2,359	2,790	1,452	350	.	.	.	.
396	8,604	.	115	1,162	2,468	2,926	1,551	382	.	.	.	.
397	9,727	.	127	1,271	2,769	3,295	1,803	462	.	.	.	.
398	10,855	.	121	1,413	3,117	3,671	2,007	526	.	.	.	.
399	11,488	.	114	1,522	3,337	3,873	2,100	542	.	.	.	.
400	10,891	.	131	1,467	3,178	3,636	1,979	500	.	.	.	.
401	8,879	.	147	1,250	2,650	2,910	1,567	355	.	.	.	.
402	7,130	.	151	1,057	2,200	2,290	1,204	228	.	.	.	.
403	6,618	.	145	1,017	2,071	2,100	1,093	192	.	.	.	.
404	6,643	.	151	1,029	2,074	2,104	1,093	192	.	.	.	.
405	5,778	.	155	912	1,799	1,809	939	164	.	.	.	.
406	4,662	.	154	708	1,377	1,432	827	164	.	.	.	.
407	5,682	.	160	667	1,485	1,800	1,266	304	.	.	.	.
408	7,358	.	162	751	1,851	2,385	1,765	444	.	.	.	.
409	7,924	.	151	800	2,017	2,593	1,891	472	.	.	.	.

Continued on next page

Table 3 – continued from previous page

Round	sum	1	2	3	4	5	6	7	8	9	10	11
410	8,039	.	151	811	2,061	2,639	1,905	472	.	.	.	.
411	8,634	.	154	815	2,182	2,854	2,101	528	.	.	.	.
412	9,878	.	159	863	2,444	3,283	2,489	640	.	.	.	.
413	10,531	.	153	902	2,592	3,503	2,685	696	.	.	.	.
414	11,126	.	143	907	2,611	3,621	2,896	892	56	.	.	.
415	12,978	.	141	956	2,792	4,070	3,515	1,336	168	.	.	.
416	14,946	.	127	1,060	3,139	4,636	4,116	1,644	224	.	.	.
417	15,760	.	119	1,166	3,355	4,881	4,315	1,700	224	.	.	.
418	15,859	.	102	1,305	3,683	4,962	4,073	1,538	196	.	.	.
419	16,847	.	83	1,562	4,367	5,505	3,884	1,297	149	.	.	.
420	19,430	.	76	1,879	5,226	6,702	4,262	1,187	98	.	.	.
421	22,250	.	69	2,040	5,922	7,810	4,912	1,360	133	4	.	.
422	26,575	.	65	2,244	6,591	8,992	6,121	2,166	378	18	.	.
423	32,536	.	77	2,430	7,279	10,491	7,981	3,471	765	42	.	.
424	37,681	.	84	2,572	7,944	11,933	9,588	4,467	1,035	58	.	.
425	40,997	.	87	2,697	8,609	13,096	10,480	4,847	1,119	62	.	.
426	42,391	.	87	2,820	8,999	13,616	10,764	4,916	1,127	62	.	.
427	43,025	.	84	2,975	9,259	13,798	10,804	4,916	1,127	62	.	.
428	44,104	.	77	3,013	9,360	14,023	11,140	5,171	1,242	78	.	.
429	46,352	.	76	3,055	9,590	14,575	11,836	5,654	1,456	110	.	.
430	45,641	.	76	2,892	9,124	13,955	11,643	5,925	1,765	245	16	.
431	42,639	.	72	2,333	7,471	11,783	10,964	6,667	2,657	632	60	.
432	50,108	.	71	1,810	6,525	12,066	13,739	9,747	4,642	1,320	188	.
433	65,663	.	66	1,521	6,773	14,770	19,071	14,182	6,990	1,970	320	.
434	73,800	.	70	1,614	7,463	17,174	22,064	15,920	7,382	1,849	264	.
435	72,340	.	53	1,613	7,607	17,556	21,923	15,185	6,660	1,535	208	.
436	74,086	.	36	1,431	7,489	17,668	22,754	15,896	6,996	1,596	220	.
437	90,529	.	23	1,361	7,897	20,452	27,861	20,700	9,533	2,366	336	.
438	111,375	.	22	1,409	9,080	24,916	34,474	25,938	12,073	3,035	428	.
439	134,302	.	26	1,674	10,749	29,387	41,016	31,596	15,086	4,081	659	28
440	170,820	.	24	1,910	11,979	34,283	50,293	41,713	21,936	7,081	1,457	144
441	215,812	.	22	1,984	13,178	39,988	62,251	54,876	30,442	10,570	2,265	236
442	252,203	.	24	1,952	14,055	45,042	72,702	65,772	36,918	12,818	2,656	264
443	257,132	.	25	2,000	14,300	46,033	74,596	67,334	37,404	12,684	2,516	240
444	199,296	.	24	2,009	12,509	37,198	57,654	51,007	27,737	9,197	1,789	172
445	121,891	.	23	1,983	10,040	25,116	34,924	29,299	15,052	4,608	778	68
446	73,540	.	23	2,042	8,511	17,585	20,606	15,487	7,169	1,879	222	16