Cube Attacks on Tweakable Black Box Polynomials

Itai Dinur and Adi Shamir

Computer Science department
The Weizmann Institute
Rehobot 76100, Israel

Abstract. Almost any cryptographic scheme can be described by *tweakable polynomials* over $GF(2)$, which contain both secret variables (e.g., key bits) and public variables (e.g., plaintext bits or IV bits). The cryptanalyst is allowed to tweak the polynomials by choosing arbitrary values for the public variables, and his goal is to solve the resultant system of polynomial equations in terms of their common secret variables. In this paper we develop a new technique (called a *cube attack*) for solving such tweakable polynomials, which is a major improvement over several previously published attacks of the same type. For example, on the stream cipher Trivium with a reduced number of initialization rounds, the best previous attack (due to Fischer, Khazaei, and Meier) requires a barely practical complexity of 2^{55} to attack 672 initialization rounds, whereas a cube attack can find the complete key of the same variant in 2^{19} bit operations (which take less than a second on a single PC). Trivium with 735 initialization rounds (which could not be attacked by any previous technique) can now be broken with 2^{30} bit operations. Trivium with 767 initialization rounds can now be broken with 2^{45} bit operations, and the complexity of the attack can almost certainly be further reduced to about 2^{36} bit operations. Whereas previous attacks were heuristic, had to be adapted to each cryptosystem, had no general complexity bounds, and were not expected to succeed on random looking polynomials, cube attacks are provably successful when applied to random polynomials of degree d over n secret variables whenever the number m of public variables exceeds $d + log_d n$. Their complexity is $2^{d-1}n + n^2$ bit operations, which is polynomial in n and amazingly low when d is small. Cube attacks can be applied to any block cipher, stream cipher, or MAC which is provided as a black box (even when nothing is known about its internal structure) as long as at least one output bit can be represented by (an unknown) polynomial of relatively low degree in the secret and public variables.

Keywords: Cryptanalysis, algebraic attacks, cube attacks, tweakable black box polynomials, stream ciphers, Trivium.

1 Introduction

Solving large systems of multivariate polynomial equations is considered an exceedingly difficult problem, which had been studied extensively over many years.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-3-642-01001-9_35

A. Joux (Ed.): EUROCRYPT 2009, LNCS 5479, pp. 278–299, 2009.

The problem is NP-complete even when the system contains only quadratic equations modulo 2 (see [18]), and it provides the main protective mechanism in many cryptographic schemes.

The main mathematical tool developed in order to solve such equations is the notion of Grobner bases (see [1],[2] and [3]), but when we try to apply it in practice to random equations with more than 100 variables it usually runs out of space without providing any answers. The much simpler linearization technique considers each term in these polynomials as a new independent variable, and tries to solve the resultant system of linear equations by Gauss elimination. Its main problem is that it requires a hugely overdefined system of polynomial equations. For example, a system of 256 polynomial equations of degree $d = 16$ in $n = 256$ variables over $GF(2)$ is expected to have a unique solution, but in order to find it by linearization we have to increase the number of equations to the number of possible terms in these equations, which is about $n^d = 2^{128}$. There are several improved algorithms such as XL and XSL (see [3],[4],[5], [6] and [7]) which reduce the number of required equations and the time and space complexities, but they are still completely impractical for such sizes.

The main observation in this paper is that the polynomial equations defined by many cryptographic schemes are not arbitrary and unrelated. Instead, they are typically variants derived from a single *master polynomial* by setting some *tweakable variables* to any desired value by the attacker. For example, in block ciphers and message authentication codes (MAC's) the output depends on key bits which are secret and fixed, and on message bits which are public and controllable by the attacker in a chosen plaintext attack. Similarly, in stream ciphers the output depends on secret fixed key bits and on public IV bits which can be chosen arbitrarily. By modifying the values of these tweakable public bits, the attacker can obtain many *derived polynomial equations* which are closely related. What we show in this paper is that when the master polynomial is sufficiently random, we can eliminate with *provably high probability* all of its n^d nonlinear terms by considering a surprisingly small number of only $2^d n$ tweaked variants, and then solve a precomputed version of the resultant n linear equations in n variables using only n^2 bit operations. For example, when $d = 16$ and $n = 10,000$, we can simultaneously eliminate all the 2^{200} nonlinear terms by considering only the 2^{20} derived polynomial equations obtained by encrypting 2^{20} chosen plaintexts defined by setting 20 public bits to all their possible values. After this "massacre" of nonlinear terms, the only thing left is a random looking system of linear equations in all the secret variables, which is easy to solve. In case the master polynomial is not random, there are no guarantees about the success rate of the attack, and if the degree of the master polynomial is too high, the basic attack technique is not likely to work. For these cases, we describe in the appendix several generalizations which may prove to be useful.

To demonstrate the attack, consider the following dense master polynomial of degree $d = 3$ over three secret variables x_1, x_2, x_3 and three public variables v_1, v_2, v_3:

$$P(v_1,v_2,v_3,x_1,x_2,x_3) = v_1v_2v_3 + v_1v_2x_1 + v_1v_3x_1 + v_2v_3x_1 + v_1v_2x_3 + v_1v_3x_2 + v_2v_3x_2 + v_1v_3x_3 + v_1x_1x_3 + v_3x_2x_3 + x_1x_2x_3 + v_1v_2 + v_1x_3 + v_3x_1 + x_1x_2 + x_2x_3 + x_2 + v_1 + v_3 + 1$$

Third degree polynomials over six variables can have $\binom{6}{3}+\binom{6}{2}+\binom{6}{1}+\binom{6}{0}=42$ possible terms, and thus there are 2^{42} such polynomials over $GF(2)$. To eliminate all the 35 possible nonlinear terms by Gauss elimination, we typically need 35 such polynomials. By setting the three public variables v_1, v_2, v_3 to all their possible 0/1 values, we can get only 8 derived polynomials, which seem to be insufficient. However, summing the 4 derived polynomials with $v_1 = 0$ we get $x_1 + x_2$, summing the 4 derived polynomials with $v_2 = 0$ we get $x_1 + x_2 + x_3$, and summing the four derived polynomials with $v_3 = 0$ we get $x_1 + x_3$, which simultaneously eliminated all the nonlinear terms. When we numerically sum modulo 2 the values of the derived polynomials in these three different ways (instead of symbolically summing the polynomials themselves), we get a simple system of three linear equations in the three secret variables. Consequently, the master nonlinear polynomial can be solved by a chosen message attack which evaluates it for just 8 combinations of values of its public variables.

Since we deal with dense multivariate polynomials of relatively high degree, their explicit representations are extremely big, and thus we assume that they are provided only implicitly as black boxes which can be queried. This is a natural assumption in cryptanalysis, in which the attacker can interact with an encryption black box that contains the secret key. A surprising consequence of our approach is that we can now attack completely unknown cryptosystems (such as the CRYPTO-1 algorithm implemented in millions of transportation smart cards, whose design was kept as a trade secret until very recently) which are embedded in tamper resistant hardware, without going through the tedious and expensive process of physical reverse engineering! Since the number of queries we use is much smaller than the number needed in order to uniquely interpolate the polynomial from its black box representation, our algorithm manages to break such unknown cryptosystems even when it is information theoretically impossible to uniquely determine them from the available data.

Some of the issues we deal with in this paper are how to efficiently estimate the degree d of a given black box multivariate polynomial, how to solve high degree polynomials which can be well approximated by low degree polynomials (e.g., when they only contain a small number of high degree terms which almost always evaluate to zero), and how to easily find the linear equations defined by the sums of these huge derived polynomials. Note that in the black box model the attacker is not allowed to perform symbolic operations such as asking for the coefficient of a particular term, evaluating the GCD of two polynomials, or computing their Grobner basis, unless he first interpolates them from their values by a very expensive procedure which requires a huge number of queries.

We call this cryptanalytic technique a *cube attack* since it sets some public variables to all their possible values in n (not necessarily disjoint) $(d-1)$-dimensional boolean cubes, and sums the results in each cube. The attack is not completely new, since some of its ideas and techniques were also used in previous heuristic attacks on various cryptosystems, but we believe that this is the first time that all these elements were brought together, accompanied by careful analysis of their complexity and success rate for random black box polynomials.

Cube attacks should not be confused with the *interpolation attacks* of Jakobsen and Knudsen ([17]), which deal with cryptosystems whose basic operations are quadratic polynomials over all or half of the input. Such polynomials are univariate or bivariate polynomials over $GF(2^n)$, and thus have fairly compact representations which can be easily interpolated from sufficiently many input/output pairs. Our attack deals with huge black box multivariate polynomials over $GF(2)$ which cannot possibly be interpolated from the available data.

The attack is remotely related to the *square attack* (see [8]) which considers the special case of cryptographic schemes whose secret bits are grouped into longer words, which are arranged in a two dimensional square. Cube attacks make no such assumptions about how the secret bits in the polynomial equations are related to each other, and thus they can be applied in a much broader set of circumstances.

The attack is also superficially similar to *integral attack* (also called *saturation attack* in the literature) and to *high order differential* attack which sum the output of cryptosystems over various subsets of input variables. However, as explained in section 3, this is just an artifact of the special field $GF(2)$ in which addition and subtraction are the same operation, and over a general field $GF(p^k)$ with $p > 2$ we have to use a different way to apply cube attacks.

Several previously published techniques try to break particular schemes by highly heuristic attacks that sum output values on some Boolean cubes of public variables. These related attacks include [26], [27], [28], [29], [30] and [31], and are collectively referred to as *chosen IV statistical attacks.* Compared to these attacks, the cube attack is much more general, is applicable to block ciphers in addition to stream ciphers, and has a better-defined preprocessing phase which does not need adaptations for each given scheme. As a result, cube attacks can be applied with provable success rate and complexity even when the cryptosystem is modelled by a random black box polynomial about which nothing is known. The most important difference is that in cube attacks each summation leads to an easily solvable linear equation (in any number of secret key bits), whereas in chosen IV statistical techniques there are many attack scenarios, and each summation typically leads only to a statistically biased expression (in a small subset of the secret key bits). Such a bias has to be amplified by many repetitions using a much larger amount of data before it can be used in order to find the key. The most convincing demonstration of this difference is the best previously known chosen IV attack on the Trivium stream cipher [28]: When the number of initialization rounds is reduced to 672, this attack has a relatively high complexity of 2^{55} operations, whereas the standard unoptimized cube attack can perform full key recovery in just 2^{19} bit operations; When the number of initialization steps is increased to 735, no previously published attack is faster than exhaustive search, whereas the same cube attack can easily perform full key recovery in 2^{30} bit operations. These and further results about Trivium are discussed in the appendix.

2 Terminology

This section describes the formal notation we use in the rest of the paper. The attacker is given a black box that evaluates an unknown polynomial p over $GF(2)$ of $n+m$ inputs bits $(x_1, .., x_n, v_1, .., v_m)$ and outputs a single bit. The polynomial is assumed to be in Algebraic Normal Form, namely, the sum of products of variables. The input bits $x_1, .., x_n$ are the secret variables, while $v_1, .., v_m$ are the public variables. The solution consists of two phases. During the preprocessing phase, the attacker is allowed to set the values of all the variables $(x_1, .., x_n, v_1, .., v_m)$ and to use the black box in order to evaluate the corresponding output bit of p. This corresponds to the usual cryptanalytic setting in which the attacker can study the cryptosystem by running it with various keys and plaintexts. During the online phase, the n secret variables are set to unknown values, and the attacker is allowed to set the values of the m public variables $(v_1, .., v_m)$ to any desired values and to evaluate p on the combined input.

To simplify our notation, we ignore in the rest of this section the distinction between secret and public variables, and denote all of them by $x_1, ..., x_n$. Since $x_i^2 = x_i$ modulo 2, the terms t_I in the polynomial can be indexed by the subset $I \subseteq \{1, ..., n\}$ of the variables which are multiplied together, and every polynomial can be represented by sums of t_I for a certain collection of subsets I. We denote by $\mathbb{P}_d^n$ the set of all the multivariate polynomials over $GF(2)$ with n variables and total degree bounded by d.

Given a multivariate polynomial p and any index subset I, we can factor the common subterm t_I out of some of the terms in p, and represent the polynomial as the sum of terms which are supersets of I and terms which are not supersets of I:

$$p(x_1, .., x_n) \equiv t_I \cdot p_{S(I)} + q(x_1, .., x_n)$$

We call $p_{S(I)}$ the *superpoly* of I in p. Note that for any p and I, the superpoly of I in p is a polynomial that does not contain any common variable with t_I, and each term in $q(x_1, .., x_n)$ misses at least one variable from I.

To demonstrate these notions, let

$$p(x_1, x_2, x_3, x_4, x_5) = x_1x_2x_3 + x_1x_2x_4 + x_2x_4x_5 + x_1x_2 + x_2 + x_3x_5 + x_5 + 1$$

be a polynomial of degree 3 in 5 variables, and let $I = \{1, 2\}$ be an index subset of size 2. We can represent p as:

$$p(x_1, x_2, x_3, x_4, x_5) = x_1x_2(x_3 + x_4 + 1) + (x_2x_4x_5 + x_3x_5 + x_2 + x_5 + 1)$$

where

$$\begin{aligned} t_I &= x_1x_2 \\ p_{S(I)} &= x_3 + x_4 + 1 \\ q(x_1, x_2, x_3, x_4, x_5) &= x_2x_4x_5 + x_3x_5 + x_2 + x_5 + 1 \end{aligned}$$

Definition 1. *A* maxterm *of p is a term t_I such that $deg(p_{S(I)}) \equiv 1$, i.e. the superpoly of I in p is a linear polynomial which is not a constant.*

Any subset I of size k defines a k-dimensional Boolean cube of 2^k vectors C_I in which we assign all the possible combinations of 0/1 values to variables in I, and leave all the other variables undetermined. Any vector $\boldsymbol{v} \in C_I$ defines a new derived polynomial $p_{|\boldsymbol{v}}$ with $n-k$ variables (whose degree may be the same or lower than the degree of the original polynomial). Summing these derived polynomials over all the 2^k possible vectors in C_I, we end up with a new polynomial, which is denoted by $p_I \triangleq \sum_{\boldsymbol{v} \in C_I} p_{|\boldsymbol{v}}$. In the next section, we prove that this polynomial has a simple alternative definition, which makes it extremely useful in cryptanalytic applications.

3 The Main Observation

Theorem 1. *For any polynomial p and subset of variables I, $p_I \equiv p_{S(I)}$ modulo 2.*

Proof. Write $p(x_1,..,x_n) \equiv t_I \cdot p_{S(I)} + q(x_1,..,x_n)$. We first examine an arbitrary term t_J of $q(x_1,..,x_n)$, where J is the subset containing the variable indexes that are multiplied together in t_J. Since t_J misses at least one of the variables in I, it is added an even number of times (for the two possible values of any one of the missed variables, where all the other values of the variables are kept the same), which cancels it out modulo 2 in $\sum_{\boldsymbol{v} \in C} p_{|\boldsymbol{v}}$.

Next, we examine the polynomial $t_I \cdot p_{S(I)}$: All $\boldsymbol{v} \in C_I$ zero t_I, except when we assign the value 1 to all the variables in I. This implies that the polynomial $p_{S(I)}$ (which has no variables with indexes in I and is thus independent of the values we sum over) is summed only once, when t_I is set to 1. Consequently, the formal sum of all the derived polynomials is exactly the superpoly $p_{S(I)}$ of the term we sum over. □

Basically, the theorem states that the sum of the 2^k polynomials derived from the original polynomial p by assigning all the possible values to the k variables in I, eliminates all the terms except those which are contained in the superpoly of I in p. The summation thus reduces the total degree of the master polynomial by at least k, and if t_I is any maxterm in p, this sum yields a linear equation in the remaining variables. For example, if we sum the polynomial $p(x_1, x_2, x_3, x_4, x_5)$ defined in the previous section over the four possible values of x_1 and x_2 in the maxterm $t_I = x_1 x_2$, we get the linear expression $p_{S(I)} = (x_3 + x_4 + 1)$. Consequently, all the cryptanalyst has to do in order to solve a tweakable master polynomial of degree d is to find sufficiently many maxterms in it, and for each maxterm to sum at most 2^{d-1} derived polynomials. Note that he only has to add the 0/1 *values* of these derived polynomials (which he can obtain via a chosen plaintext attack), and not their huge symbolic expressions. The summed bit is then equated with a fixed linear expression which can be derived from the master black box polynomial during a separate preprocessing stage, since it is not key-dependent. For low degrees such as $d = 16$, the derivation of the right hand side of each linear equation during the online phase of the attack requires at most $2^{15} = 32768$ additions of single bit values, which takes a negligible amount of time.

Over a general field $GF(p^k)$ with $p > 2$, the correct way to apply cube attacks is to alternately add and subtract the outputs of the master polynomial with public inputs that range only over the two values 0 and 1 (and not over all their possible values of $0, 1, 2, ..., p-1$), where the sign is determined by the sum (modulo 2) of the vector of assigned values. In this form, they are reminiscent of FFT computations. Cube attacks are thus more closely related to high order differential attacks than to integral attacks, but they do not use the same formal operator. For example, consider the bivariate polynomial $p(x, v) = 4x^2v^3 + 3x^2v^5 \pmod 7$ of degree 7. The formal derivative of this polynomial with respect to v is the 6-degree polynomial $p'_v(x, v) = 5x^2v^2 + x^2v^4 \pmod 7$ whereas our numeric difference yields $p(x, 1) - p(x, 0) = (4x^2 + 3x^2) - (0 + 0) = 0 \pmod 7$ which has degree 0. In addition, cube attacks use algebraic rather than statistical techniques to actually find the secret key.

4 The Preprocessing Phase

Given an explicit description of the master polynomial, it is easy to split it into $p(x_1, .., x_n) \equiv t_I \cdot p_{S(I)} + q(x_1, .., x_n)$ for any term t_I. However, when the exponentially long master polynomial is given only as a black box, it is not clear how to find this representation, and how to store it in a compact way.

When t_I is a maxterm, the issue of compact representation becomes easy, since we only have to know its superpoly $p_{S(I)}$ in order to apply the attack, and this expression is a short linear combination of some of the secret variables x_i, with the possible addition of the constant 1. Note that we can eliminate all the public variables v_i that are not summed over from this linear expression by fixing each one of them to 0 (or to 1) during the summation.

In order to actually find $p_{S(I)}$ for a given black box master polynomial and a maxterm t_I in it, we use a separate preprocessing phase in which the attacker is given the extra power of tweaking both the public and the secret variables:

Theorem 2. *Let t_I be a maxterm in a black box polynomial p. Then:*

1. *The free term in $p_{S(I)}$ can be computed by summing modulo 2 the values of p over all the inputs of $n + m$ variables which are zero everywhere except on the $d - 1$ variables in the summation cube C_I.*
2. *The coefficient of x_j in the linear expression $p_{S(I)}$ can be computed by summing modulo 2 all the values of p for input vectors which are zero everywhere except on the summation cube C_I and all the values of p for input vectors which are zero everywhere except on the summation cube and at x_j which is set to 1.*

The proof is based on the observation that in a linear expression, the coefficient of any variable x_j is 1 if and only if flipping the value of x_j flips the value of the expression, and the free term can be computed by setting all the variables to zero.

In the rest of this section, we distinguish between the cases of random and non-random master polynomials.

4.1 Preprocessing Random Polynomials

In many cryptographic schemes, the mixing of the inputs is so thorough that the representation of each ciphertext bit as a fully expanded polynomial function of the n key bits and m plaintext bits can be viewed as a random polynomial:

Definition 2. *A* random polynomial *of degree d in $n+m$ variables is a polynomial $p \in \mathbb{P}_d^{n+m}$ such that each possible term of degree at most d is independently chosen to occur with probability* 0.5*.*

In fact, the notion of randomness we need in order to lower bound the success probability of cube attacks is considerably weaker, since the only terms which play any role in the attack are those that correspond to maxterms in p:

Definition 3. *A d-*random *polynomial with $n+m$ variables is a polynomial $p \in \mathbb{P}_d^{n+m}$ such that each possible term of degree d which contains one secret variable and $d-1$ public variables is independently chosen to occur with probability* 0.5*, and all the other terms can be chosen arbitrarily.*

In any d-random polynomial, any term t_I which is the product of $d-1$ public variables v_i has an extremely high probability to be a maxterm: Its corresponding superpoly is a polynomial of degree at most 1, and it is a polynomial of degree 0 only when for all the secret variables x_i the terms $t_I x_i$ are not chosen to appear in the polynomial. The probability of this event is 2^{-n}.

For any two terms t_{I_1} and t_{I_2} which are the products of $d-1$ public variables, we get independent random choices of their corresponding superpolys, even when I_1 and I_2 are almost identical. For example, when $d=4$, $I_1=\{1,2,3\}$, and $I_2=\{1,2,4\}$, each one of the two terms $v_1v_2v_3x_5$ and $v_1v_2v_4x_5$ occurs in p with probability 0.5 independently of the other. Since we do not need disjoint subsets of public variables as our maxterms, we only need about $d+\log_d n$ tweakable public variables in order to pack n different maxterms among their products, since $\binom{d+\log_d n}{d} = \binom{d+\log_d n}{\log_d n} \approx d^{\log_d n} = n$. In particular, when $d=16$ and $n=10,000$, it suffices to have only $m=20$ tweakable public variables to apply the cube attack, since $\binom{20}{15} = 15,504 > n$. Note that the *computations* of these maxterms are not independent since we reuse the same derived polynomials in many overlapping cube summations, but the *results* of the computations are independent linear combinations of the secret variables.

After choosing n random maxterms, the attacker defines an $n \times n$ matrix A whose rows contain their corresponding superpolys. If the matrix is nonsingular, the attacker precomputes and stores A^{-1} in order to reduce the complexity of the linear algebra in the online phase of the attack from $O(n^3)$ to $O(n^2)$.

Since A is a random matrix in which each entry is independently selected with probability 1/2, it is very easy to compute the probability that it is nonsingular:

Lemma 1. *The probability that an $n \times n$ random binary matrix over $GF(2)$ is invertible is* $\prod_{i=1}^{n}(1-2^{-i}) \approx 0.28879$

Proof. The proof is by a simple induction on the rows of the matrix.

This is a constant probability, which can be made arbitrarily close to 1 during the preprocessing phase by considering a few extra maxterms. For $d = 16$ $n = 10,000$ and $m = 20$, there are $15,504$ possible superpolys to choose from, and the probability that the rank of all these random linear expressions will be smaller than $10,000$ is negligible.

Since the preprocessing phase has to be executed only once for each cryptosystem whereas the online phase has to be executed once for each key, some cryptanalytic attacks "cheat" by allowing extremely expensive operations during an unbounded preprocessing phase which make the whole attack impractical. When cube attacks are applied to random polynomials, the complexity of the preprocessing phase is at most n times larger than that of the online phase of the attack, and thus if one phase is practically feasible so is the other.

4.2 Preprocessing Nonrandom Polynomials

When the polynomial representation of the cryptosystem is not assumed to be d-random, there are no guarantees about the success rate of the attack. The basic questions we are faced with in this case are how to estimate the degree d of the polynomial p which is only given as a black box, and how to choose appropriate maxterms if they exist. We propose the following technique, which is a variant of the random walk proposed in [28].

The attacker randomly chooses a size k between 1 and m and a subset I of k public variables, and computes the value of the superpoly of I by numerically summing over the cube C_I (setting each one of the other public variables to a static value, usually to zero). If his subset I is too large, the sum will be a constant value (regardless of the choice of secret variables), and in this case he has to drop one of the public variables from I and repeat the process. If his subset I is too small, the corresponding $p_{S(I)}$ is likely to be a nonlinear function in the secret variables, and in this case he has to add a public variable to I and repeat the process. The correct choice of I is the borderline between these cases, and if it does not exist the attacker can restart with a different initial I.

The best way to understand this process is to think about a (not necessarily random) polynomial p in which all the terms have the same degree d, but contain different proportions of secret and public variables. When we sum over subsets I with $d-2$ public variables, we will get a purely quadratic polynomial in the secret variables which corresponds to all those terms that contain the $d-2$ variables in I as their public variables and two additional secret variables. Linear terms will not occur in this polynomial since every term which contains $d-1$ public variables is eliminated by at least one public variable which is not in I and is thus set to zero. Note that for nonrandom polynomials, this quadratic expression may be empty for some I (misleading us to believe that I is too large), but nonempty for another I (indicating correctly that it is too small), and thus we may have to restart the preprocessing with several initial I's. When we sum over subsets I with $d-1$ public variables, we will get a linear polynomial in the secret variables, but again it may be empty. In particular, if all the terms in the nonrandom p contain at least two secret variables, we will never be able to get any linear superpoly during the preprocessing phase, regardless of the choice of I. When we sum over I with d public variables, we will get a key-independent constant, which is zero or one depending on whether the unique term which is the product of all the public variables in I does or does not occur in p. In this case we will always act correctly by reducing the size of I. Finally, when we sum over an I of size $d+1$ or larger, we will always get the zero polynomial, since every term in p misses at least one of the public variables in I, and will thus be added an even number of times modulo 2.

For any choice of values for all the secret variables, we sum the 0/1 values of p over the subcube C_I of public variables, setting all the other public variables to zero. This sum is a function of secret variables only, and we can test it for linearity during the preprocessing phase (in which we are allowed to modify the secret variables) by using any one of the efficient linearity tests which were developed as part of the PCP theorem (see [9]).

One example of such a linearity test is the BLR test (see [10]), which chooses vectors $\boldsymbol{x}, \boldsymbol{y} \in \{0,1\}^n$ independently and uniformly at random, and verifies that $p_{S(I)}[\mathbf{0}] + p_{S(I)}[\boldsymbol{x}] + p_{S(I)}[\boldsymbol{y}] = p_{S(I)}[\boldsymbol{x}+\boldsymbol{y}]$. The test ensures that if $p_{S(I)}$ is linear, the test always succeeds, whereas if $p_{S(I)}$ is far from being linear, the test fails with high probability. The test is repeated sufficiently many times until the attacker is convinced that $p_{S(I)}$ is very close to being linear (e.g., it it linear, except for a few high degree terms which almost always evaluate to zero). By using the cube attack in this case, we can find most but not all of the possible keys, which is good enough in our cryptanalytic application. Note that in our preprocessing, almost all the functions we test are likely to be nonlinear superpolys (which typically fail in one of the first few linearity tests, thus requiring only a few cube summations) or easily detected constant functions, whereas in the preprocessing done by Fischer Khazaei and Meier, almost all the functions they test are balanced, and distinguishing them from slightly biased functions requires a huge number of cube summations on average.

As in the random setting, the attacker stops when sufficiently many linearly independent vectors are derived and A^{-1} can be computed. The online phase of the attack is identical to the case of random polynomials.

There are many possible optimizations of this process. For example, summing the values of p over subcubes with large intersections can be sped up by memorizing various partial sums, and thus we do not have to start from scratch when we add or eliminate one public variable from I in our proposed random walk search technique. Another extension uses the freedom to choose the values of the public variables that are not summed over. In case we get an empty superpoly for a specific cube, and a non-linear superpoly for any of its sub-cubes, we can still try to make the superpoly nonempty in order to get a maxterm by setting some of the remaining public variables to one. If the result is still zero, we can set some more of these variables to one. If the result is non-linear, we can set a few public variables that are not summed over back to zero. Note that this random walk over the *values* of the public variables we do not sum over is different from the previously described random walk over the *subset* of the public variables we sum over.

A different attack scenario on non random polynomials uses the cube attack as a distinguisher rather than as a key extraction procedure. For example, if some output bit is a polynomial of degree at most d in the $n+m$ input variables, summing it over any d-dimensional cube of public variables will always give a constant value (which depends on the summation set I, but not on the key, and thus can be precomputed in the preprocessing phase), whereas in a random cipher such a sum will be uniformly distributed. Since the attacker has to sum over a single cube and does not have to solve any equations, the complexity of this distinguisher is just 2^d. Consequently, ANY cryptographic scheme in which $d < n$ and $d < m$ can be distinguished from a random cipher by an algorithm which is faster than exhaustive search, regardless of whether its polynomial representation is random or not. A detailed description of the theory and applications of cube distinguishers appearers in [19].

5 Applications to Block Ciphers

In chosen plaintext attacks on block ciphers, the public variables are the bits of the plaintext. Since most block ciphers have a block size of at least 128 bits, there is no shortage of tweakable variables.

Since the attack is using only a single bit from the ciphertext, it makes no difference whether the cryptographic mapping is invertible or not. Consequently, we can attack a keyed hash function (also known as a MAC, or message authentication code) by using exactly the same techniques. An example of such an attack on the keyed hash function MD6 can be found in [19].

The main problem in applying the cube attack to block ciphers is that they usually contain many rounds, and the degree of the polynomial grows exponentially with the number of rounds (until it hits the maximum possible value of $n+m$). Several techniques that may help to overcome the problem of high degree polynomials in block ciphers appear in the appendix.

6 Applications to Stream Ciphers

In the case of stream ciphers, the secret variables represent the key, and the public variables represent the IV. The model assumes that the attacker can simulate the cipher during the preprocessing phase, and can apply a chosen IV attack during the online phase. Note that we can also use a known IV attack if the stream cipher operates in the common counter mode that uses consecutive binary numbers (such as the packet number or the time of day) as its IV's, since their least significant bits contain full subcubes of various dimensions.

Many proposed stream ciphers use one or more linear feedback shift registers (LFSR), which are either filtered or combined by nonlinear functions to produce the output. In this case, the degree of the output polynomial is only determined by this function, is relatively small, is easy to bound, and does not increase when the cipher generates a large number of bits (many of which are kept hidden during the initialization phase). The attack requires the knowledge of only one output bit for several IV values, and we can choose its location arbitrarily. In particular, we can choose a bit location in which the corresponding plaintext bit is known. Typical examples of such locations include standard packet header bits, or the high bits of ASCII characters which are known to be zero.

As an extreme example of the power of cube attacks, consider a long LFSR with $10,000$ bits and a secret dense feedback polynomial, which is filtered by a layer of $1,000$ S-boxes. Each S-box is a different secret mapping of 8 bits from the LFSR into one output bit, and the connection pattern between the LFSR and the S-boxes is also assumed to be secret. In each clock cycle, the cipher outputs only one bit, which is the XOR of the outputs of all the S-boxes. Each bit in the LFSR is initialized by a different secret dense quadratic polynomial in $10,000$ key and IV bits. The LFSR is clocked a large and secret number of times without producing any outputs, and then only the first output bit for any given IV is made available to the attacker.

The attack is a *structural attack* which is based only on the general form of the cryptosystem (as described in figure 1). Note that the attacker does not know the secret LFSR feedback polynomial, the $1,000$ S-boxes, the LFSR/S-Box interconnection pattern, the actual key/IV mixing function, or the number of dummy initialization steps. The only properties of this design which are exploited by the cube attack are that the output of each S-box is a random looking polynomial of degree 16 (obtained by substituting quadratic expressions in each one of its 8 input variables), that the XOR of these S-boxes is also a polynomial of degree 16 (in the $10,000$ secret and public variables), and that we have sufficient tweaking power over the generation of the first output bit. The attack uses only 2^{20} output bits (one for each IV value), which are summed in $10,000$ overlapping 15 dimensional cubes (note that $\binom{20}{15} = 15504 > 10000$). The attacker can thus get $10,000$ linear equations in $10,000$ variables, which he can easily solve by using the precomputed inverse of the coefficient matrix. This stream cipher can thus be broken in less than 2^{30} bit operations, even though it could not be attacked by any previous technique, including correlation attacks or the analysis of low Hamming weight LFSR modifications (see for instance [11],[12],[13],[14],[15], and [16]).

We have experimentally tested the cube attack on this stream cipher, in order to rule out the possibility that the black box polynomials which represent this stream cipher have some unexpected properties that foil the attack. In all our tests, the attack behaved exactly as expected under the assumption that the polynomials are d-random.

Some stream ciphers such as LILI and A5/1 use clock control in order to foil correlation attacks. If A5/1 had used its clock control only when producing the output bits (but not during the initialization rounds), it would have been trivial to break it with a straightforward cube attack, which uses only the first output bit produced for each IV value.

Other types of stream ciphers such as Trivium (see [21]) include a small amount of nonlinearity in the feedback of the shift register, and thus the degree

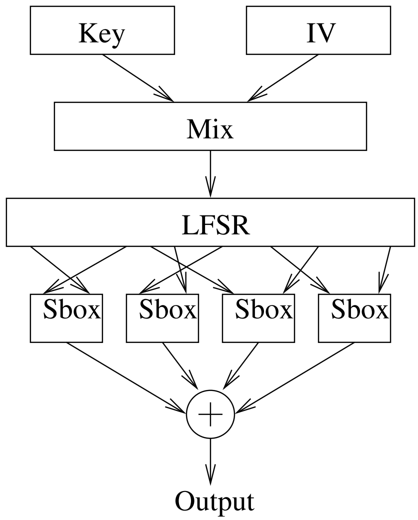

Fig. 1. A typical Filtered LFSR generator

of the output polynomial grows slowly over time. Since the attacker needs only the first output bit for each IV, it may be possible to apply the cube attack to such schemes, provided that they do not apply too many initialization rounds in which no output is produced. Results of the attack on simplified variants of Trivium that apply fewer initialization rounds are given in appendix B.

If the attacker is given more than one output bit in each execution of the stream cipher, he can slightly reduce the number of public variables required in the attack by summing the outputs of several polynomials p_i defining different output bits. This way he can get more than one linear equation for each maxterm during the preprocessing phase, and thus he can use fewer tweakable bits and use a smaller number of expensive restarts (which use many initialization steps) of the stream cipher during his attack.

An interesting observation is that unlike the case of other attacks, XOR'ing the outputs of several completely unrelated stream ciphers does not provide enhanced protection against cube attacks: If each one of the stream ciphers can be represented by a low degree multivariate polynomial, their XOR is also a low degree polynomial which can be attacked just as easily as the individual stream ciphers.

7 Conclusions

In this paper we introduced a new type of cryptanalytic attack and described some of its applications. It joins the rank of linear, differential, algebraic, and correlation attacks by being a generic attack that can be applied to many types of cryptographic schemes. We demonstrated its effectiveness by breaking (both in theory and with an actual implementation) a standard construction of a stream cipher which seems to be secure against all the previously known attacks. We also used the attack to break simplified Trivium variants with complexity that is considerably lower than the complexity of previous known attacks. The attack is likely to be the starting point for a new area of research, and hopefully it will lead to a better understanding of what makes cryptosystems secure.

Acknowledgements. We would like to thank Shahram Khazaei, Willi Meier and Paul Crowley for independently verifying our results.

References

1. Ajwa, I.A., Liu, Z., Wang, P.S.: Gröbner bases algorithm. Technical report, ICM Technical Reports Series (ICM-199502-00) (1995)
2. Faugère, J.c.: A new efficient algorithm for computing gröbner bases (f4). Journal of Pure and Applied Algebra, 75–83 (1999)
3. Gwenole, A., Jean-Charles, F., Hideki, I., Mitsuru, K., Makoto, S.: Comparison Between XL and Groebner Basis Algorithms (2004)
4. Courtois, N.T., Klimov, A.B., Patarin, J., Shamir, A.: Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 392–407. Springer, Heidelberg (2000)
5. Courtois, N., Patarin, J.: About the xl algorithm over gf(2). In: CT-RSA, pp. 141–157 (2003)
6. Yang, B.-Y., Chen, J.-M., Courtois, N.T.: On asymptotic security estimates in XL and gröbner bases-related algebraic cryptanalysis. In: López, J., Qing, S., Okamoto, E. (eds.) ICICS 2004. LNCS, vol. 3269, pp. 401–413. Springer, Heidelberg (2004)
7. Courtois, N.T., Pieprzyk, J.: Cryptanalysis of block ciphers with overdefined systems of equations. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 267–287. Springer, Heidelberg (2002)
8. Daemen, J., Knudsen, L.R., Rijmen, V.: The block cipher square. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 149–165. Springer, Heidelberg (1997)
9. Arora, S.: Probabilistic checking of proofs: a new characterization of np. Journal of the ACM, 2–13 (1998)
10. Blum, M., Luby, M., Rubinfeld, R.: Self-testing/correcting with applications to numerical problems. Journal of Computer and System Sciences 47, 549–595 (1993)
11. Courtois, N.T., Meier, W.: Algebraic attacks on stream ciphers with linear feedback, pp. 345–359. Springer, Heidelberg (2003)
12. Golic, J.D.: On the security of nonlinear filter generators. In: Proceedings of the Third International Workshop on Fast Software Encryption, London, UK, pp. 173–188. Springer, Heidelberg (1996)
13. Courtois, N.T.: Fast algebraic attacks on stream ciphers with linear feedback. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 176–194. Springer, Heidelberg (2003)
14. Englund, H., Johansson, T.: A new simple technique to attack filter generators and related ciphers. In: Selected Areas in Cryptography, pp. 39–53 (2004)
15. Golic, J.D., Clark, A., Dawson, E.: Generalized inversion attack on nonlinear filter generators. IEEE Trans. Comput. 49(10), 1100–1109 (2000)
16. Johansson, T., Jnsson, F.: Fast correlation attacks through reconstruction of linear polynomials, pp. 300–315. Springer, Heidelberg (2000)

17. Jakobsen, T., Knudsen, L.R.: The interpolation attack on block ciphers. In: Fast Software Encryption, pp. 28–40. Springer, Heidelberg (1997)
18. Garey, M.R., Johnson, D.S.: Computers, and Interactibility. A guide to the theory of np-completeness. Bell Telephone Labratories, Incorporated
19. Aumasson, J.-P., Dinur, I., Meier, W., Shamir, A.: Cube Testers and Key Recovery Attacks On Reduced-Round MD6 and Trivium. In: Fast Software Encryption. Springer, Heidelberg (2009)
20. estream: Ecrypt stream cipher project, `http://www.ecrypt.eu.org/stream/`
21. De Cannière, C., Preneel, B.: Trivium - a stream cipher construction inspired by block cipher design principles. estream, ecrypt stream cipher. Technical report, of Lecture Notes in Computer Science
22. Raddum, H.: Cryptanalytic results on trivium. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/039, 2006 (2006), `www.ecrypt.eu.org/stream/papersdir/2006/039.ps`
23. Maximov, A., Biryukov, A.: Two trivial attacks on trivium. In: Selected Areas in Cryptography, pp. 36–55 (2007)
24. McDonald, C.C.C., Pieprzyk, J.: Attacking bivium with minisat, `http://eprint.iacr.org/2007/040`
25. Sönmez Turan, M., Kara, O.: Linear approximations for 2-round trivium. In: Proc. First International Conference on Security of Information and Networks (SIN 2007), pp. 96–105. Trafford Publishing (2007)
26. Englund, H., Johansson, T., Sönmez Turan, M.: A framework for chosen IV statistical analysis of stream ciphers. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT 2007. LNCS, vol. 4859, pp. 268–281. Springer, Heidelberg (2007)
27. Vielhaber, M.: Breaking one.fivium by aida an algebraic iv differential attack. Cryptology ePrint Archive, Report 2007/413
28. Fischer, S., Khazaei, S., Meier, W.: Chosen IV statistical analysis for key recovery attacks on stream ciphers. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 236–245. Springer, Heidelberg (2008)
29. Joux, A., Muller, F.: A chosen iv attack against turing. In: Selected Areas in Cryptography, pp. 194–207 (2003)
30. O'Neil, S.: Algebraic structure defectoscopy. Cryptology ePrint Archive, Report 2007/378
31. Juhani, M., Saarinen, O.: Chosen-iv statistical attacks on estream ciphers. In: Proceeding of SECRYPT 2006, pp. 260–266 (2006)

A Appendix: Extensions and Generalizations of Cube Attacks

Various generalizations of the cube attack can be successfully applied even to cryptosystems in which the attacker cannot find sufficiently many linear superpolys, and thus the original attack fails:

1. In block ciphers, the attacker can try to use a "meet in the middle" attack. Each bit in the middle of the encryption process can be described as either a polynomial in the plaintext and key bits, or as a polynomial in the ciphertext and key bits. Since the number of rounds is halved, the degree of each one of these polynomials may be the square root of the degree of the full polynomial

which describes the cipher (especially when the number of rounds is relatively small and these degrees did not hit their maximal possible values). Instead of equating the given ciphertext bits to their high degree polynomials, the attacker can equate the two low degree polynomials describing the two halves of the encryption and get an easier to solve master equation. This technique can also be extended to the case of double encryptions, where the attacker has the additional benefit that the secret key bits used in the two polynomials are disjoint. Note that the attacker can get multiple polynomial equations for each one of the bits in the middle or for any one of their polynomial combinations.

2. In some stream ciphers with many initialization rounds, it is difficult to find the low degree maxterms required for the attack. In these cases, given that the internal structure of the stream cipher in known, we can try a different approach: The attacker explicitly represents the state register bits as polynomials in terms of the public and private variables at some intermediate initialization round. Given this explicit representation, the attacker performs linearization on the private variables by replacing them with a new set of private variables, reducing the degrees of the state register bit polynomials. The values of the new set of private variables can then be recovered using the basic techniques of the cube attack. After the values of the new private variables are recovered, the attacker can solve for the original key by solving the equations obtained during linearization. If the cipher's state is invertible, or close to being invertible, the attacker can simply run the cipher backwards to recover the key, instead of solving equations. Note that a similar technique may also be used to attack block ciphers, given that the attacker can explicitly represent the polynomials at some intermediate encryption round.
3. The attacker can benefit from any system of linear equations (even if it has fewer than n equations), or from any system of nonlinear equations in which some of the variables occur linearly, by enumerating and testing only their smaller set of solutions.
4. The attacker can exploit ANY nonlinear superpoly he can find and compactly represent by guessing some of the secret variables in it and simplifying the result. In particular, guessing $n-1$ key bits will always suffice to turn any superpoly into an easy to solve linear equation in the remaining variable, and will thus result in an attack which is faster than exhaustive search, assuming that the evaluation of the superpoly is not too time consuming.
5. The attacker can try to solve the equations he can derive from the cube attack even when they are nonlinear, provided that their degrees are low enough. When m is large, the attacker can sum over many possible subsets of $d-1$ public variables, and get a highly overdefined system of nonlinear equations which might be solved by linearization or any other technique.
6. The attacker can easily recognize quadratic superpolys by a generalization of the BLR linearity test: The attacker randomly chooses vectors $\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3} \in \{0,1\}^n$, and verifies that $p_{S(I)}[\mathbf{0}]+p_{S(I)}[\boldsymbol{x_1}]+p_{S(I)}[\boldsymbol{x_2}]+p_{S(I)}[\boldsymbol{x_3}]+p_{S(I)}[\boldsymbol{x_1}+\boldsymbol{x_2}] + p_{S(I)}[\boldsymbol{x_1}+\boldsymbol{x_3}] + p_{S(I)}[\boldsymbol{x_2}+\boldsymbol{x_3}] + p_{S(I)}[\boldsymbol{x_1}+\boldsymbol{x_2}+\boldsymbol{x_3}] = 0$. Again, non quadratic functions are likely to be eliminated after a few tests. The test can

be further generalized to cubic functions and to polynomials of higher degree with the number of required function evaluations growing exponentially with the degree. The coefficient calculation for polynomials of higher degree can be generalized as well.

7. The attacker can use the cube attack even if he cannot compactly represent superpolys. In this case, the attacker decides on a subkey (i.e. a subset of private variables) whose value is guessed during the online phase. For each value of the subkey bits, the degree of the superpolys in the remaining private variables is likely to be reduced, and the attacker can compute and store them more efficiently. Since the cubes and corresponding superpolys are now key-dependant, they need to be computed and stored for each potential value of the subkey. This requires more preprocessing time and memory, but gives the attacker the extra flexibility of using different maxterms for each subset of keys.
8. The attacker is usually given more than one output bit, and thus more than one polynomial in the input bits. In addition to trying each one of them separately, he can test any polynomial combination of these polynomials and try to find some linear superpolys among these combinations.
9. Note that in the common mode of operation of stream ciphers in which $n = m$, and the secret key and public IV bits are XOR'ed together during the initialization step, the maximal possible degree of the polynomial representation of the scheme is n, whereas in the general case the maximal possible degree is $n + m$.
10. When the cryptographic scheme has an insufficient number of public variables (or none at all), we can recast the cube attack as a related key attack in which we are also allowed to flip some of the secret key bits during the online phase. By replacing some of the x_i variables by the combinations $x_i + v_i$, we may get linear $p_{S(I)}$ polynomials where none existed before.

B Appendix: Cube Attacks on Scaled-Down Trivium Variants

Trivium [21] is a stream cipher designed in 2005 by C. De Canni‘ere and B. Preneel and submitted to the Profile 2 (hardware) European project eSTREAM [20]. It has an exceptionally simple structure, which leads to very good performance in both hardware and software. Despite Trivium's simplicity, there are no substantial cryptanalytic results against it so far. Due to these outstanding qualities, Trivium was chosen as part of the portfolio for Profile 2 by the eSTREAM project.

B.1 Description of Trivium

Trivium's internal state consists of 288 bits stored in three NLFSRs of different lengths. In each round, each register is shifted by one bit. The feedback to each register consists of a non linear combination of bits from another register, XORed with a bit from the same register. The output bit at the end of each round is a linear combination of six state bits, two taken from each register. During initialization, the 80-bit key is placed in the first register, and the 80-bit IV is placed in the second register. The other state bits are set to zero, except the last three bits in the third register, which are set to one. The state is then updated $4 \times 288 = 1152$ times without producing an output.

B.2 Previous Attacks

Trivium has a simple structure which led many cryptanalysts to try to attack it. Nevertheless, to this day, there are no attacks better than exhaustive search on the full version of Trivium. Due to Trivium's cryptanalytic resistance, scaled-down variants have been proposed and studied by cryptanalysts hoping to better understand the full-scale version. Two scaled-down variants named Bivium A and Bivium B were introduced in [22]. Both of these variants have an internal state composed of only 2 shift registers. Previous attacks on Trivium and its Bivium variants are summarized below:

– Raddum [22] developed an algorithm for solving sparse quadratic equations. The algorithm was used to break Bivium A in "about a day", and requires 2^{56} seconds to break Bivium B. The complexity of the attack applied to Trivium is 2^{164}.
– Maximov and Biryukov [23] developed a technique that can be applied to Bivium and Trivium. The technique involves guessing certain key bits and key bit products that reduce the Trivium quadratic equation system to a linear equation system that can be solved by linear algebra. The technique can be used to recover the state of Bivium B with complexity of $c \cdot 2^{36.1}$, and to recover the state of Trivium with complexity of $c \cdot 2^{83.5}$, where the constant c is the complexity of solving the system of linear equations.
– McDonald, Charnes, and Pieprzyk [24] showed that the MiniSat algorithm can be used to attack Bivium B with complexity of about 2^{56}.

Another family of scaled-down Trivium variants, assumes that fewer than 1152 initialization rounds are performed before producing an output. Previous attacks on Trivium variants with fewer than 1152 initialization rounds are summarized below:

– Turan and Kara [25] used linear cryptanalysis to give a linear approximation with bias 2^{-31} for Trivium with 288 initialization rounds.
– Englund, Johansson, and Turan [26] developed statistical tests and used them to show statistical weaknesses of Trivium with up to 736 initialization rounds. The basic idea is to use a statistical (rather than algebraic) variant of a cube attack, which selects an IV subset, examines all the keystream produced by assigning this subset all possible values, while keeping the other IV bits fixed. The key stream is viewed as a function of the selected IV subset variables, and statistical tests are performed to distinguish this function from a random one.

– Vielhaber [27] recovered 47 key bits of Trivium with 576 initialization rounds in negligible time. The key bits were recovered after some small IV special subsets were found, each one with the following property: The result of summing on some keystream bit produced by assigning a special subset all possible IV values, while keeping the other IV bits fixed, is equal to either one of the key bits or to the sum of two key bits. Note that this is a very special case of our cube attack, and it is not clear why the author imposed this unnecessary restriction.
– Fischer, Khazaei, and Meier [28] combined statistical tests with the method described in [27], and showed an attack on Trivium with 672 initialization rounds with complexity 2^{55}.

The last three attacks share their cube summing element with our attack, but then proceed in a different way, which does not apply efficient linearity testing to the resultant superpolys in order to find easy to solve linear equations. Our greatly improved cryptanalytic results for Trivium clearly demonstrate that cube attacks are more general, more efficient, and more powerful than these previous techniques.

B.3 The Attack

We summarize the results we obtained so far for various simplified variants of Trivium. All the maxterms and their associated linear equations were obtained by running the preprocessing phase of the cube attack in a high level language on a single PC over several weeks, and much better results can be expected by using a more optimized implementation on a cluster of more powerful computers.

– The best known attack on the variant which uses 672 initialization rounds is described by Fischer, Khazaei, and Meier in [28]. The authors attack this variant with complexity 2^{55}. We were able to find 63 linearly independent maxterms during the preprocessing phase of the cube attack on this variant (in fact, we found more, but the additional maxterms do not reduce the total complexity of the attack). All of the maxterms correspond to cubes of size 12. The maxterms and cubes are listed in Table 1 next to the summed output bit index. Both the key bit indexes and the IV bit indexes range from 0 to 79. The output bit index ranges from 672 to 685, hence the attacker needs up to 14 initial output bits produced by the cipher after the 672 key mixing rounds. Each of the maxterms passed at least 100 linearity tests, and thus the maxterm equations are likely to be correct for most keys. During the online phase of the cube attack, the attacker has to find the values of the linear equations defined by these maxterms by summing over the 63 cubes, of size 12. This requires a total of about 2^{18} chosen IVs. After the maxterm values are computed, the rest of the key can be recovered by exhaustive search with complexity 2^{17}. The total complexity of the attack is thus no more than 2^{19}, which is a big improvement compared to the best known attack. Note that the maxterms are very sparse, hence the complexity of the linear algebra in the preprocessing and online phases is negligible.

- We pushed the attack further by strengthening the Trivium variant to use 735 initialization rounds before producing an output. Currently, there is no known attack that is better than exhaustive search on this scaled-down Trivium variant. We were able to find 53 linearly independent maxterms corresponding to cubes of size 23 (again, we have more). The total complexity of the online phase of the attack is less than 2^{30}, which is much better than exhaustive search.
- Pushing the attack even further, we were able to find so far 35 maxterms for the stronger Trivium variant that uses 767 initialization rounds. The maxterms are listed in Table 2 in the appendix, next to the corresponding cubes. Most cubes are of size 29, but there are a few cubes of size ranging from 28 to 31. The complexity of the attack is 2^{45} since it is dominated by an exhaustive search for the $80 - 35 = 45$ missing key bits, after the values of the linear equations defined by these maxterms are computed. Computation on weaker variants shows that once a cube of a certain size that corresponds to a maxterm is found, we can expect to find many more cubes of the same size with linear superpolys. Thus, given more preprocessing resources, it is very likely that the online phase complexity of the attack can be reduced to about 2^{36}.

Our results show that even after many key mixing initializations rounds, Trivium is still breakable with complexity that is significantly faster than exhaustive search. We are still investigating the resistance of stronger Trivium variants to cube attacks and their generalizations.

B.4 Details of the New Cube Attacks on Scaled-Down Trivium Variants

Tables 1 and 2, list the maxterms, cube IV indexes, and output bit indexes for Trivium with 672 and with 767 initialization rounds respectively. In each one of the summations in Table 1 , all the public variables that do not belong to the cube were set to 0. In a few summations in Table 2, some public variables that do not belong to the cube were set to 1. These are specified in the last column. IV and key bits are indexed as in the original Trivium specification starting from 0 to 79 (e.g. key bits 65 and 68 and IV bits 68 and 77 determine the output bit with index 0).

Table 1. Maxterms for Trivium with 672 Initialization rounds

Maxterm Equation	Cube Indexes	Output Bit Index
1+x0+x9+x50	{2,13,20,24,37,42,43,46,53,55,57,67}	675
1+x0+x24	{2,12,17,25,37,39,46,48,54,56,65,78}	673
1+x1+x10+x51	{3,14,21,25,38,43,44,47,54,56,58,68}	674
1+x1+x25	{3,13,18,26,38,40,47,49,55,57,66,79}	672
1+x2+x34+x62	{0,5,7,18,21,32,38,43,59,67,73,78}	678
1+x3+x35+x63	{1,6,8,19,22,33,39,44,60,68,74,79}	677
x4	{11,18,20,33,45,47,53,60,61,63,69,78}	675
x5	{5,14,16,18,27,31,37,43,48,55,63,78}	677
x7	{1,3,6,7,12,18,22,38,47,58,67,74}	675
1+x8+x49+x68	{1,12,19,23,36,41,42,45,52,54,56,66}	676
x11	{0,4,9,11,22,24,27,29,44,46,51,76}	684
x12	{0,5,8,11,13,21,22,26,36,38,53,79}	673
x13	{0,5,8,11,13,22,26,36,37,38,53,79}	673
1+x14	{2,5,7,10,14,24,27,39,49,56,57,61}	672
x15	{0,2,9,11,13,37,44,47,49,68,74,78}	685
x16	{1,6,7,12,18,21,29,33,34,45,49,70}	675
x17	{8,11,15,17,26,23,32,42,51,62,64,79}	677
x18	{0,10,16,19,28,31,43,50,53,66,69,79}	676
x19	{4,9,10,15,21,24,32,36,37,48,52,73}	672
x20	{7,10,18,20,23,25,31,45,53,63,71,78}	675
1+x20+x50	{11,16,20,22,35,43,46,51,55,58,62,63}	675
1+x21+x66	{10,13,15,17,30,37,39,42,47,57,73,79}	673
x22	{2,4,21,23,25,41,44,54,58,66,73,78}	673
x23	{3,6,14,21,23,27,32,40,54,57,70,71}	672
1+x24	{3,5,14,16,18,20,33,56,57,65,73,75}	672
1+x28	{6,11,14,19,33,39,44,52,58,60,74,79}	676
x29	{1,7,12,18,21,25,29,45,46,61,68,70}	675
x30	{2,8,13,19,22,26,30,46,47,62,69,71}	674
x31	{3,9,14,20,23,27,31,47,48,63,70,72}	673
x32	{4,10,15,21,24,28,32,48,49,64,71,73}	672
x33	{2,4,6,12,23,29,32,37,46,49,52,76}	680
1+x34+x62	{0,5,7,13,18,21,32,38,43,59,73,78}	678
1+x35+x63	{1,6,8,14,19,22,33,39,44,60,74,79}	677
x36	{2,4,5,8,15,19,27,32,35,57,71,78}	677
x38+x56	{0,3,4,9,20,28,33,41,54,58,72,79}	678
1+x39+x57+x66	{8,11,13,17,23,25,35,45,47,54,70,79}	674
x40+x58+x64	{0,6,10,16,19,31,43,50,66,69,77,79}	676
1+x41	{2,15,17,20,21,37,39,44,46,56,67,73}	674
x42+x60	{1,16,20,22,34,37,38,53,58,69,71,78}	674
x43	{2,7,14,22,41,45,48,58,68,70,72,76}	673
x44+x62	{3,14,16,18,20,23,32,46,56,57,65,73}	672
1+x45+x64	{0,6,10,16,18,28,31,43,53,69,77,79}	676
x46+x55	{2,8,11,13,28,31,35,37,49,51,68,78}	684
x47	{5,8,20,32,36,39,45,51,65,69,76,78}	676
x48	{2,4,10,14,16,22,25,44,49,51,57,78}	678
x49+x62	{1,12,19,23,36,41,42,45,52,56,69,75}	676
x51+x62	{1,7,8,13,21,23,28,30,47,68,71,75}	674
x52	{5,8,9,12,16,18,23,40,44,63,66,70}	674
x53	{2,11,21,24,32,55,57,60,63,66,70,77}	675
1+x54+x60	{4,7,10,18,20,25,50,53,61,63,71,78}	675
x55+x64	{5,12,16,19,22,36,47,55,63,71,77,79}	674
1+x56	{4,9,18,21,23,27,32,38,43,58,67,69}	677
x57	{1,7,9,14,18,21,33,40,45,49,59,68}	675
1+x58	{2,6,12,13,19,23,30,48,55,59,69,79}	673
x60	{5,7,10,13,15,17,28,40,47,73,76,79}	681
x61	{13,21,24,39,42,46,48,51,55,61,72,78}	673
1+x62	{2,4,10,11,19,34,47,55,56,58,69,77}	674
x63	{5,7,10,15,17,35,40,47,52,57,76,79}	674
x64	{8,11,13,17,23,25,35,47,62,64,68,79}	673
x65	{2,3,13,15,19,29,32,37,39,51,76,79}	682
1+x66	{5,7,10,13,15,17,35,40,52,70,76,79}	678
1+x67	{5,20,24,29,33,35,37,39,63,65,74,78}	677
1+x68	{1,12,19,23,36,41,52,54,56,66,69,75}	676

Table 2. Maxterms for Trivium with 767 Initialization rounds

Maxterm Equation	Cube Indexes	Output	Variables set to 1
1+x0	{1,3,4,7,9,10,12,16,19,21,25,27,29,30,32,34,35,37,40,47,50,51,60,61,64,67,72,73,79}	769	
x3	{0,3,6,9,12,15,18,21,24,27,30,32,37,40,43,44,48,50,53,57,59,61,63,64,66,68,71,73,77,79}	773	{11}
x20	{1,3,5,7,10,14,18,20,22,23,26,30,36,38,42,43,44,45,47,49,52,54,60,63,69,71,72,73,78}	770	{53}
x22	{1,3,5,7,10,12,14,16,18,20,23,26,30,39,41,42,43,47,50,52,53,55,58,60,61,64,69,71,78}	769	
x23	{0,2,4,6,8,10,14,17,19,21,23,26,30,34,35,36,43,45,46,48,49,54,59,64,67,72,73,74,75,79}	767	
1+x29	{1,3,5,7,10,12,14,17,20,22,24,30,32,34,37,38,40,41,48,50,54,56,58,59,65,66,68,70,78}	774	
x30	{0,2,4,6,8,10,14,17,19,21,23,26,30,33,34,35,36,43,45,46,49,54,57,59,62,64,72,73,75,79}	773	{67}
1+x31	{0,2,4,6,8,10,13,14,17,19,21,23,26,30,31,34,35,36,37,42,53,60,61,64,66,69,72,73,77,79}	773	
x32	{0,2,4,6,8,10,14,17,19,21,23,25,26,27,30,32,34,43,44,53,58,63,68,70,71,72,75,78,79}	772	{33,37,38}
1+x33+x60+x66+x68	{1,3,5,7,10,14,18,20,23,26,30,35,37,39,40,41,44,48,49,51,54,58,59,60,61,64,70,75,77,78}	772	
1+x34	{1,3,5,7,10,12,14,16,17,20,24,28,30,33,34,36,40,42,45,46,51,52,54,56,62,66,70,77,78}	770	{76}
x35	{1,3,4,6,7,8,9,12,14,16,19,21,25,27,30,38,41,44,45,48,50,55,57,60,63,65,71,73,79}	769	
x36	{0,2,4,5,6,8,10,14,17,19,21,23,26,27,30,37,39,40,47,48,55,62,65,70,73,75,77,78,79}	768	{54}
x37	{1,3,5,7,10,12,14,16,17,20,24,26,30,32,35,37,41,45,46,54,58,60,64,67,68,69,70,72,78}	770	
x38	{0,2,4,6,8,10,14,17,19,23,25,26,30,34,36,38,40,42,44,53,56,57,60,63,69,72,73,75,79}	768	{39}
x41	{0,1,3,4,7,10,12,15,17,19,22,24,25,28,30,34,39,42,44,52,56,58,59,62,64,68,70,72,79}	773	{71}
1+x45	{1,3,5,7,10,12,14,16,18,20,22,23,26,30,33,39,42,43,47,50,52,53,55,58,60,64,71,77,78}	769	
1+x46	{1,3,5,8,11,14,16,17,19,21,23,26,27,29,30,32,36,38,42,44,45,49,51,53,59,60,63,64,75,76,78}	771	
x51	{0,2,4,6,8,10,14,17,19,23,26,30,33,38,39,41,43,46,47,50,54,58,59,60,62,63,64,71,72,77,79}	773	
1+x53+x57	{1,3,5,7,10,14,16,18,20,23,26,30,35,37,39,41,44,48,49,51,54,58,60,64,68,70,75,77,78}	773	{40,61}
x54	{0,2,4,6,8,10,14,17,19,23,26,30,33,38,39,41,43,46,50,54,59,60,61,62,63,64,70,74,77,79}	767	
1+x55	{1,3,5,7,10,12,14,17,18,20,24,27,30,33,36,38,40,41,44,53,56,59,61,66,68,72,75,76,78}	771	
x56	{1,3,5,7,9,12,14,16,19,21,23,25,27,30,35,37,40,51,56,62,63,64,67,69,71,74,75,76,79}	769	
1+x57	{1,3,5,7,10,12,14,17,20,24,30,32,34,37,38,40,48,50,52,54,56,57,58,59,63,66,68,70,78}	774	
x58	{0,2,4,6,8,10,14,17,19,21,23,26,30,33,36,43,45,48,49,54,57,59,62,64,67,72,74,75,79}	767	
x59+x65	{1,3,5,7,10,12,14,17,20,22,24,26,28,30,35,40,41,42,44,52,54,60,65,67,68,73,74,75,78}	773	
x60	{2,4,10,13,15,19,23,25,27,31,33,34,37,40,41,45,48,50,51,54,56,60,61,62,67,69,71,73,76}	770	
1+x60+x66	{1,3,4,5,7,9,12,16,19,21,25,27,30,32,33,35,38,40,43,45,47,51,55,57,59,60,62,75,79}	774	
x61	{3,5,11,14,16,20,24,26,28,32,34,35,38,41,42,46,49,51,52,55,57,61,62,63,68,70,72,74,77}	769	
x62	{1,3,5,7,10,12,14,17,20,22,24,26,28,30,35,40,41,42,44,47,52,54,65,66,67,68,73,75,78}	772	
1+x62+x68	{1,3,5,7,10,12,14,17,20,22,24,26,28,30,35,40,41,42,44,47,52,59,60,67,68,73,75,77,78}	773	
x63	{2,4,8,10,13,15,19,23,27,31,33,37,40,41,45,48,50,54,56,60,61,62,67,69,71,73,76,78}	770	
x64	{3,5,9,11,14,16,20,24,28,32,34,38,41,42,46,49,51,55,57,61,62,63,68,70,72,74,77,79}	769	
x65	{0,2,4,6,7,8,10,14,17,19,21,23,26,30,32,34,36,37,39,41,43,45,55,56,61,66,74,76,79}	767	
1+x67	{2,4,6,8,11,13,15,17,19,21,23,24,27,31,34,40,42,43,44,48,51,56,59,61,65,70,72,78,79}	768	