Improving Key Recovery to 784 and 799 Rounds of Trivium Using Optimized Cube Attacks

Pierre-Alain Fouque[1(✉)] and Thomas Vannet[2]

[1] Rennes 1 University, Rennes, France
pierre-alain.fouque@ens.fr
[2] NTT Secure Platform Laboratories, Tokyo, Japan
thomas.vannet@lab.ntt.co.jp

Abstract. Dinur and Shamir have described cube attacks at EUROCRYPT '09 and they have shown how efficient they are on the stream cipher Trivium up to 767 rounds. These attacks have been extended to distinguishers but since this seminal work, no better results on the complexity of key recovery attacks on Trivium have been presented. It appears that the time complexity to compute cubes is expensive and the discovery of linear superpoly also requires the computation of many cubes. In this paper, we increase the number of attacked initialization rounds by improving the time complexity of computing cube and we show attacks that go beyond this bound. We were able to find linear superpoly up to 784 rounds, which leads to an attack requiring 2^{39} queries. Using quadratic superpoly, we were also able to provide another attack up to 799 rounds which complexity is 2^{40} queries and 2^{62} for the exhaustive search part. To achieve such results, we find a way to reduce the density of the polynomials, we look for quadratic relations and we extensively use the Moebius transform to speed up computations for various purposes.

Keywords: Trivium · Cube attacks · Cryptanalysis · Moebius transform

1 Introduction

After every stream cipher submitted to the NESSIE project in 2000 was successfully broken, a new European project called eSTREAM was started in 2004 in order to build new secure stream ciphers. One of the ciphers submitted, Trivium, has a very simple design and yet no attack was discovered on its full version, which uses 1152 initialization rounds.

There have been various attempts to attack reduced variants of Trivium. In [12], Vielhaber managed to recover 47 bits of the key after 576 rounds using an algebraic method. Afterwards, Dinur and Shamir described a full key recovery in less than 2^{30} requests to Trivium limited to 735 rounds and recovered 35 key bits after 767 rounds in about 2^{36} requests using the so-called cube attacks in [7].

S. Moriai (Ed.): FSE 2013, LNCS 8424, pp. 502–517, 2014.
DOI: 10.1007/978-3-662-43933-3_26,

Consequently, the on-line attack requires an exhaustive search of 2^{45}. With the introduction of cube testers, Aumasson *et al.* were able to build a distinguisher on Trivium after 790 rounds and detect non-randomness properties after up to 885 rounds in [2]. Independently, in [10], Knellwolf, Meier and Naya-Plasencia built a distinguisher up to 806 rounds for all keys and up to 961 rounds for some very specific keys.

Despite a warning suggesting that the cube attack may be applied on stronger versions of Trivium, no such results were obtained since 2009. In this paper, we investigate how far the cube attack can realistically be applied to Trivium. Cube attacks usually are composed of an online and an offline phase. During the offline phase, the adversary is looking for linear relations and once the number of such relations are high enough, the on-line phase requires to compute cube using chosen public variables. In [7], only the complexity of the online phase is given. The only information regarding the offline phase is that it lasted "several weeks" for the strongest version, i.e. the attack on 767 rounds of Trivium. However, this phase of the attack is the most interesting if we try to extend the previous attacks. We were able to reproduce this phase in a matter of hours and a stronger version in a couple of days worth of computation. The online phase has a low-enough complexity to be feasible in practice.

Our Contributions. First of all, we develop efficient implementations and optimizations to compute large cubes and for instance, we can compute a cube of size 27 in less than a second on a standard computer. Using fast implementation of the Moebius Transform, we realize many interesting computations such as polynomial density measurements, degree testing or interpolation. In addition, we used a different method than the one suggested in [7] to test many different parameters at once. Thanks to this, over 90,000 computations which should each require 2^{36} operations can be computed in merely 2 hours where this would have required about 13,000 hours with the classical method, offering an average computing time of 80ms for every cube of size 36.

Furthermore, we investigated ways to smartly select the parameters with an empirical filter which greatly improved the rate at which linear key bits relations were established. This preselection technique is the main ingredient we propose and we use to push the attack further with little increase in complexity. Indeed, we were able to recover 42 key bits after 784 rounds of initialization for the first time with less than 2^{38} cipher requests in the online phase. Adding a phase of brute-force, the 80-bit key can be fully recovered in less than 2^{39} requests. After 799 rounds, 12 key bits can be recovered directly, which leads to an attack. Besides, studying the polynomials of degree 2 found after 784 rounds, we experimentally interpolated several quadratic polynomials which reveal information for a very large amount of keys after up to 799 initialization rounds. This phase of the attack has a complexity about 2^{39}. This in turn allows a full key recovery attack in 2^{68} requests, then reduced to about 2^{62}.

Organization of the Paper. In Sect. 2, we describe the Trivium stream cipher, recall how cube attacks work and we present the Moebius transform. In Sect. 3, we show that this transformation can be used in many places to improve the complexity of Cube attacks and then, we explain how we can reduce empirically the density of the polynomial in Trivium. This technique allows to look for linear relations more easily and increases the number of initialization rounds we can attack. Finally, we present our result on Trivium using 784 rounds. In Sect. 4, we present an attack on 799 initialization rounds. To extend the attack further, we need to look for quadratic relations and we show that we have to look for specific relations, otherwise the search would not be possible. We also use the previous technique to reduce the density and then, we show that we obtain quadratic and linear relations.

2 Backgrounds

2.1 Trivium Description

The stream cipher Trivium [4] has an internal state of 288 bit registers $s_1, \ldots, s_{288}$ and works with a 80-bit key $x_1, \ldots, x_{80}$ using a 80-bit initialization vector $v_1, \ldots, v_{80}$.

It has three non-linear feedback shift registers (NLFSR) which are updated in the following way after each of the 1152 rounds of initialization:

$$
\begin{aligned}
t_1 &\leftarrow s_{66} + s_{93} \\
t_2 &\leftarrow s_{162} + s_{177} \\
t_3 &\leftarrow s_{243} + s_{288} \\
z_i &\leftarrow t_1 + t_2 + t_3 \\
t_1 &\leftarrow t_1 + s_{91} \cdot s_{92} + s_{171} \\
t_2 &\leftarrow t_2 + s_{175} \cdot s_{176} + s_{264} \\
t_3 &\leftarrow t_3 + s_{286} \cdot s_{287} + s_{69} \\
(s_1, s_2, \ldots, s_{93}) &\leftarrow (t_3, s_1, \ldots, s_{92}) \\
(s_{94}, s_{95}, \ldots, s_{177}) &\leftarrow (t_1, s_{94}, \ldots, s_{176}) \\
(s_{178}, s_{279}, \ldots, s_{288}) &\leftarrow (t_2, s_{178}, \ldots, s_{287})
\end{aligned}
$$

During the initialization phase, the output bit z_i is discarded. Once this phase is over, $(z_i)_i$ is the generated stream of bits which will be added to the plaintext (Fig. 1).

Furthermore, we denote by *recursive expression of Trivium* the following equations:

$$
\begin{cases}
s_{1,r+1} = s_{243,r} + s_{288,r} + s_{286,r} \cdot s_{287,r} + s_{69,r} \\
s_{94,r+1} = s_{66,r} + s_{93,r} + s_{91,r} \cdot s_{92,r} + s_{171,r} \\
s_{178,r+1} = s_{162,r} + s_{177,r} + s_{175,r} \cdot s_{176,r} + s_{264,r}
\end{cases}
$$

Trivium was a candidate for the hardware profile of the eSTREAM competition. As such, it is designed to be implemented with a small number of gates and was not designed to be efficiently used in software applications. Even then, the

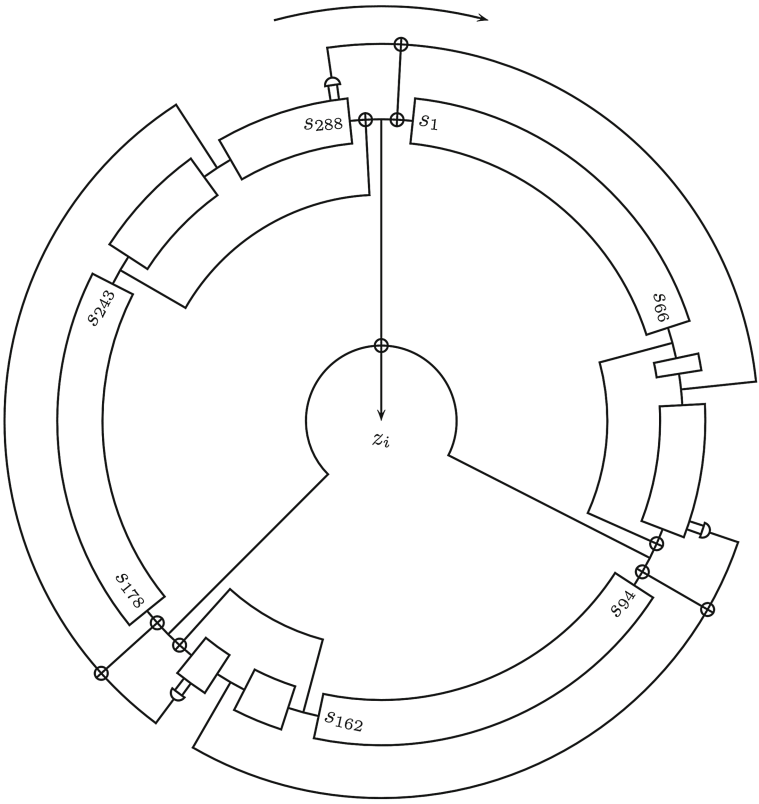

Fig. 1. The Trivium cipher.

standard software implementation is very efficient. Indeed, after being generated by the feedback function, a bit will not be used again for at least 64 rounds. This is used to compute 64 initialization rounds at once. As such, the registers only have to be updated $1152/64 = 18$ times during the initialization phase.

2.2 Cube Attacks

Given a cipher on secret bits and public bits, the cube attack, introduced by Shamir and Dinur in [7], allows one to find linear or low degree relations between key bits. Then using simple linear algebra or Grobner basis techniques, it is possible to recover the bit values. The following method is used, in particular in the easier case of linear relations:

Let $x_1, \ldots, x_n$ be the secret key bits and $v_1, \ldots, v_p$ the public bits (plaintext or initialization vector) under the attacker's control. In the ring $\mathcal{R} = \mathbb{F}_2[x_1, \ldots, x_n, v_1, \ldots, v_p]$, we consider the polynomial representation of the first output bit of the cipher as the polynomial $P(x_1, \ldots, x_n, v_1, \ldots, v_p)$ in $\mathcal{R}$. Given a subset ("cube") of the public variables $C = \{v_{c_1}, \ldots, v_{c_k}\}$ of size k, we write

P as $P = v_{c_1} \dots v_{c_k} P_C + P_R$, $P_C, P_R \in R$, where no monomial of P_R is divisible by $v_{c_1} \dots v_{c_k}$.

Then, given an assignment of the variables outside of C, summing P over every possible assignment of the variables of C will give the evaluation of the polynomial P_C on this assignment. In other words, $\sum_C P = P_C$ in $\mathcal{R}$. Indeed, every monomial of P_R will be summed with itself an even number of times in the process because at least one variable of C does not appear in said monomial. Meanwhile, $v_{c_1} \dots v_{c_k}$ will be non-zero only once, when every variable of the cube is set to 1. We call P_C the *superpoly* yielded by C and $\prod_{i \in C} v_i$ is called a *maxterm* if the superpoly yielded by C is *linear*.

Therefore, if P has a *low-enough degree* d, even though it has a large number of variables, linear relations between key bits can be obtained by summing 2^{d-1} evaluations of the cipher. Once sufficient such linear relations have been found in an offline phase which has to be done only once, the linear system can be partially solved during the online phase. All it requires is the evaluation of all the cubes obtained in the offline phase for a specific unknown key. Finally, the full key can be recovered with a phase of offline brute-force.

However, if P is a uniformly random polynomial of high degree d, then it is extremely unlikely that there exists maxterms of size $k < d - 1$. Indeed, this would require every single monomial of degree at most $d - k$ to be linear.

Because the feedback function of Trivium as described in Sect. 2.1 has a single monomial of degree 2, the polynomials in the formal representation of Trivium are expected to retain a low degree even after hundreds of initialization rounds. Furthermore, the output function of Trivium is simply the sum of the values of 6 registers, which does not affect the degree at all.

This makes reduced versions of this cipher suitable targets for the cube attack. Indeed, even though the theoretical maximum degree is 160 (since there are 160 variables), linear expressions were yielded by cubes of size only 29 after 767 rounds. In the same fashion we have found cubes of size 30 yielding linear superpolys after 784 rounds and of size 32 after 799 rounds.

2.3 Moebius Transformation

In [8], Dinur and Shamir suggest using the Moebius transform as described in [9] to compute every single subcube of a large cube at once. Because this is such a powerful tool, we describe it and show some of the ways we used it to study the cube attacks on Trivium.

Let us consider the algebraic normal form of a polynomial $P \in \mathbb{F}_2[X]$, where $X = X_1, \dots, X_n$: $P = \sum_{\sigma \in \{0,1\}^n} \alpha_\sigma X^\sigma$ with for all σ, $\alpha_\sigma \in \mathbb{F}_2$ and where $X^{\sigma_1 \dots \sigma_n} = X_1^{\sigma_1} \dots X_n^{\sigma_n}$.

Now the Moebius transform of P is the function $P^m : \begin{matrix} \{0,1\}^n \to \mathbb{F}_2 \\ \sigma \to \alpha_\sigma \end{matrix}$.

In other words, given the truth table of a boolean polynomial, the Moebius transform returns the truth table of a function indicating whether a given monomial is part of the polynomial. It is computed with the following simple algorithm (proof is provided in [9]).

The innermost loop is executed $\sum_{i=0}^{k-1} \frac{2^k}{2 \cdot 2^i} \cdot \sum_{j=0}^{2^i-1} 1 = \sum_{i=0}^{k-1} 2^{k-1} = k \cdot 2^{k-1}$ times, yet it consists of a single assignment and an exclusive or operation. Besides, the 32-bit implementation presented in [9] performs roughly 32 times less operations. Similarly, we implemented a 128-bit version with a complexity of the order of $k \cdot 2^{k-7}$ operations.

Require: Input Truth table S of Boolean function P, with 2^k entries
  Variable Sz is the small table size
  Variable Pos is the small table position
  for $i = 0$ to $k - 1$ **do**
    Let $Sz \leftarrow 2^i$, $Pos \leftarrow 0$
    while $Pos < 2^k$ **do**
      for $j = 0$ to $Sz - 1$ **do**
        $S[Pos + Sz + j] \leftarrow S[Pos + j] \oplus S[Pos + Sz + j]$
      Let $Pos \leftarrow Pos + 2 \cdot Sz$
  Output overwritten content of S containing Moebius transform

Now if we consider a cube $C = \{v_{c_1}, \ldots, v_{c_k}\}$ and the polynomial representation of a cipher $P(x_1, \ldots, x_n, v_1, \ldots, v_p) \in \mathcal{R}$, by fixing every variable outside C to some constant, we obtain a polynomial on the variables of C which we may call Q. Now $Q^m(\sigma_1, \ldots, \sigma_k)$ is the value of the superpoly yielded by $D = \{v_{c_i} | \sigma_i = 1\}$ when every variable of $C \setminus D$ is set to 0.

As it happens, a 128-bit implementation of the attack on r rounds can compute a cube of size k in $r \cdot 2^{k-7}$ calls to Trivium's round function, where $r \approx 800$ and $k \approx 40$. In this regard the computation of the Moebius transform, while still non-negligible, remained much faster than the computation of a single cube. Of course, because the full truth table has to be stored, a lot of memory may be required. Indeed, the computation of all subcubes of a cube of size k requires 2^k bits of memory.

3 An Attack on 784 Rounds of Trivium

In this section, we will describe a cube attack on Trivium up to 784 rounds. We will mainly explain how we manage to compute linear superpoly in an efficient manner.

3.1 Using the Moebius Transform to Improve the Cube Attack

In this section, we will describe how the Moebius Transform may improve the complexity of the cube search phase in the cube attacks.

Moebius Transform Is Interesting to Compute Polynomial Density. An interesting application of the Moebius transform is the ability it gives us to empirically measure the density of a black-box polynomial. Mainly we are interested in checking up to which degree the polynomial can be considered random (every monomial appears with probability 1/2). However testing the presence of a given monomial is an NP-hard problem [5]. With the Moebius transform, we can however test many monomials at once and get an approximate result with a single exponential computation.

We can choose a large cube of size k and create the truth table of the boolean function of the k variables in the cube and returning the first bit produced by the cipher. Applying the Moebius transform on this truth table, one can check how many monomials of every degree less than k appear in this restricted polynomial. Because $\binom{k}{d}$ becomes very large when d decreases (for $d \geq k/2$), this method provides an accurate result. Of course since we restrict the polynomial to a fraction of its variables (in Trivium's case, likely no more than 40 out of 160), there is no guarantee that the full polynomial P would follow the same density distribution. Still, a low density observed on monomials of a given degree on those variables is a clear indicator that the polynomial can not be considered random at this degree.

Tables 1, 2, 3 and 4 show some of the results obtained with this method for randomly selected cubes.

Table 1. Observed polynomial density after 732 rounds

Monomial size	23	24	25	26	27	28
Density (%)	41.21	30.19	16.14	5.47	1.01	0

Table 2. Observed polynomial density after 768 rounds

Monomial size	25	26	27	28	29	30	31
Density (%)	49.14	46.95	42.34	34.49	21.84	6.21	0

Table 3. Observed polynomial density after 784 rounds

Monomial size	30	31	32	33	34	35	36	37	38
Density (%)	48.74	47.09	42.38	35.57	25.38	15.07	6.90	1.60	0

Table 4. Observed polynomial density after 799 rounds

Monomial size	33	34	35	36	37	38	39
Density for random cube (%)	49.89	49.55	48.25	44.19	34.07	16.47	3.66
Density for selected cube (%)	38.44	28.36	16.82	7.31	1.84	0.15	0

As can be observed, the degree up to which the cipher behaves like a random polynomial increases very quickly until it reaches the point where neither measurements nor cube computations can be done efficiently.

Moebius Transform Is Interesting to Test Polynomial Degree. Another obvious application of the transform is to simply realize the constant /linearity /degree 2 testing required in the attack and described in Sect. 4.1 on many cubes at once. Doing this, every linearity or degree 2 test has to be fully computed on the large cube before the test is applied. Because of this, it is not possible to optimize by aborting the computations on cubes that are clearly not linear. Still, using 2^{40} bits of memory (128 GB), one can compute $\binom{40}{35} = 658,008$ cubes of size 35 and $\binom{40}{36} = 91,390$ cubes of size 36 in only 2^{40} calls to Trivium, compared to about $658,008 \cdot 2^{35} + 91,390 \cdot 2^{36} > 2^{54}$ calls with straight up cube computations. Using a computer with 16 cores running at 2.67 GHz, the full computation takes about two hours.

Because one will naturally try to use a large cube as large as can fit in the computer's memory (the method gets considerably much better results when the large cube is much larger than the target size), it is not possible to store the cube values for every key needed for the test (over 50 keys, most likely). This is not likely to be an issue as there is little reason to store the values of subcubes with a small size. For instance, when computing on a large cube of size 40 after 800 rounds, the probability of finding a linear superpoly from a subcube of size 30 or less is negligible and $\sum_{i=31}^{40} \binom{40}{i} < 2^{29} \ll 2^{40}$.

Moebius Transform Is Interesting to Interpolate Polynomials. Once a maxterm has been found, the polynomial in question has to be interpolated. For Trivium, this typically requires 80 additional cube computations. If the search for maxterms was done through a Moebius transform, it is likely several cubes have been found at once. Depending on the number of cubes found and their sizes, it can be more profitable to interpolate them all at once through yet another Moebius transform on the smallest cube containing every maxterm studied. In most cases, this will be the original large cube.

3.2 Empirical Reduction of Density

The cube attacks described in [7] rely on the fact that the studied black-box polynomial has a low degree. However, as explained previously, even though the degree of polynomials in Trivium grows slowly, it is very likely it reaches 160 by the end of the 1152 initialization rounds. Studying the size of cubes required to find linear maxterms also shows that the degree reaches well over 40 after 800 initialization rounds and is likely to reach 80 by 900 initialization rounds, at which point the attack would be no better than brute force.

However, for a sparse enough polynomial, one can hope to find linear combinations of key bits even when monomials of much higher degree remain. This

brings us to wonder how dense the polynomial representation of Trivium is. First, looking at the results presented in [7], one can see Trivium should not be considered a random polynomial. For such a polynomial, one would expect about half of the key bits to be present in linear terms, however most of them contain a single key bit, and none more than four. This leads us to believe that specific cubes of small size yielding very sparse low-degree polynomials coexist with large cubes yielding high-degree or dense polynomials. While the random walk described in [7] will detect the latter, we would be more interested in discovering the former.

To achieve this, let us look at the formal expression of the output bit of Trivium. It is the sum of 6 registers $s_{66} + s_{93} + s_{162} + s_{177} + s_{243} + s_{288}$ where each of these registers can be recursively expressed as a polynomial with a single degree 2 monomial. As such there are 12 registers involved in the high-degree part of the polynomial expression of the output bit.

Now we are looking for a maxterm c in the output polynomial P. Let us assume c has a size greater than the degree of the monomials outside the high-degree part of P. Then the polynomial yielded by c out of P is the sum of the 6 polynomials yielded by c out of the terms of degree 2, $P_{1,1}P_{1,2}, \ldots, P_{6,1}P_{6,2}$, in the recursive expression of P. If these polynomials are independent, it is unlikely their non linear monomials will cancel each other during the sum. As such, we also assume c is a maxterm for every $P_{i,1}P_{i,2}$. A sufficient condition for this property is that every partition $\{c_1, c_2\}$ of c is such that for all i the product of the polynomials yielded by c_1 out of $P_{i,1}$ and by c_2 out of $P_{i,2}$ when every variable of c is set to 1 is linear or constant.

This is the property we want to satisfy. Since in practice the maxterms tend to yield polynomials with only a couple monomials, it is to be expected that many such partitions will yield the zero polynomial on all 12 registers involved. Looking at it another way, this is an empirical reduction of the output polynomial's density by checking that its high-degree part is the product of low-density polynomials.

Once again, the search can be achieved with the Moebius transform [9]. After picking two disjoint large cubes, one can find many subcubes yielding the zero polynomial on all 12 coefficients. Now the disjoint union of two such subcubes is a candidate for the cube c.

This method does not directly produce very satisfying results, but the large number of cube candidates allows for further restrictions. The natural next step is to pick disjoint cubes c_1 and c_2 both of which have as many subcubes yielding the zero polynomial as possible. In practice, it is usually possible to find two disjoint cubes of size k such that each of their subcubes of size at least $k - 3$ yields the zero polynomial on all twelve registers involved. The union of c_1 and c_2 gives us a large cube on which to apply the Moebius transform to study its subcubes, many of which are candidates. As can be observed in Table 4, the reduction in density is notable and in return the number of maxterms detected in a single transform increased from an average of 83 to an average of 916 after

784 rounds. Similarly, after 799 rounds, the only way maxterms have been found was using this method and its drastic reduction of density.

3.3 Results on 784 Rounds

Using the heuristic explained in Sect. 3.2, notable reductions in polynomial density have been observed in practice. Mainly tests have been conducted on the version reduced to 784 rounds since a large cube of size 38 is sufficient to find linear superpoly and takes reasonable time to be computed. In such a situation, filtering the cubes only takes about 5 min while actual computation of a cube of size 38 takes about 30 min (28 h to fully complete a linearity test). Doing this, the density of monomials of size 34 falls from over 25 % to less than 10 %.

The Annex A contains the list of all 42 distinct key bits ("superpoly") which have been recovered after 784 rounds, along with the cubes yielding them. As can be observed, they are made of 30 to 33 indices and a lot of them are subsets of the same cube of size 38. One can also notice that every bit between x_{49} and x_{69} has been recovered, while none were found between x_{70} and x_{80}, suggesting that the mixing is not applied uniformly on the key. This is the same in the cubes found by Dinur and Shamir: they have all the bits between 54 and 67 and none between 70 and 80 for 735 rounds.

4 An Attack on 799 Rounds of Trivium

4.1 Testing Properties of Boolean Functions in a Black-Box Manner

Testing Constant and Linearity. The most common linearity test for polynomials is the Blum Luby Rubinfeld (BLR) test [3]. Given a black-box polynomial P on n variables one wants to test for linearity, the BLR test requires the computation on random inputs X_1 and X_2, two vectors of n bits, on the 0 vector and on $X_1 + X_2$. One then simply checks whether $P(X_1) + P(X_2) + P(0) = P(X_1 + X_2)$. The value of $P(0)$ can be computed once at the start but every subsequent test will require another 3 queries, each query in turn being the result of a cube computation in time 2^k where k is the size of the cube (in pratice, at least 30). What is more, most polynomials tested are very far from being linear and a weaker and faster version of the test could be used and still effectively detect non linear polynomials.

To achieve this, we simply pick at random r vector of bits $K_1, \ldots, K_r$ representing the keys, and test for every $r(r-1)/2$ pair of keys $\{K_i, K_j\}$ if the black-box polynomial yielded by a cube is such that $P(K_i) + P(K_j) + P(0) = P(K_i + K_j)$. This way, $r(r-1)/2$ linearity tests are done in merely $1 + r + r(r-1)/2 = r(r+1)/2 + 1$ queries instead of $1 + 3r(r-1)/2$. In practice, we chose $r = 10$ which brought the total number of queries to 56 instead of 136. To discard the non linear polynomials as soon as possible, linearity tests are conducted right when they become available. In other words, we test $P(0), P(K_1), P(K_2), P(K_1 + K_2), P(K_3), P(K_1 + K_3), P(K_2 + K_3)$, etc.

However most polynomials tested which are not of degree at least two are actually constant polynomials. As such, they would pass every single linearity test and waste a lot of time. We then arbitrarily decided to stop the tests if the polynomial returns the same value 20 times. In this case however, it is important to test all 10 different keys by the time the 20 tests are done. Otherwise, only 6 different random keys would be tested and with probability 1/64 a linear, non constant superpoly would be discarded. Since we hope to find about 80 such superpolys, 10 independent random keys seem sufficient.

It is worth noting that there exist other linearity tests besides BLR which can detect with greater precision if a polynomial is not quite linear. However they are usually based on machine learning techniques and require a number of queries too high to be applied to cube attacks where a single query can take several hours of computation. Besides, an almost-linear superpoly is sufficient for the attack as it will return the correct value of a key bit in almost every case.

Testing Degree 2. The special case of testing for quadratic polynomials is a very interesting one. First, it is not very costly. The classical test consists in independently picking 3 random keys k_1, k_2 and k_3 and checking whether $P(0)+P(k_1)+P(k_2)+P(k_3)+P(k_1+k_2)+P(k_1+k_3)+P(k_2+k_3)+P(k_1+k_2+k_3) = 0$. However, because we have already queried on every sum of 2 random keys while testing the polynomial for linearity, the only unknown expression is $P(k_1 + k_2 + k_3)$. Thus we get a test for degree 2 with a single extra cube computation over the linearity test. Since the keys are not chosen independently, the test is a bit weaker than the theoretical one, but once again it is sufficient to discard most polynomials of higher degree.

In practice, with r initially picked random keys, it is then theoretically possible to realize $r(r-1)(r-2)/6$ tests of degree 2. For $r = 10$, this is 120 tests. Because this would be quite long, we decided to stop after 56 tests, mimicking the number of queries for the linearity test. In addition, polynomials of degree 2 are easy to exploit to directly obtain key bits values. Because of the general sparseness of the polynomials interpolated we can hope to find isolated monomials of degree 2 among monomials of lower degree. This would reveal two keys bits for a fourth of all keys, which is non negligible. Furthermore, if one of the two bits happens to be known through a linear expression, the value of the other one can be deduced for half of all possible keys.

Besides, when particular shapes arise (see Sect. 4.2 for details), it can become easy to find additional linear relations between the key bits. For instance, if $x_i \cdot x_{i+1} + x_{i+2} = \alpha$ and $x_{i+1} \cdot x_{i+2} + x_{i+3} = \beta$, $\alpha, \beta \in \mathbb{F}_2$, then $x_{i+1} \cdot x_i + \alpha x_{i+1} + x_{i+3} = \beta$ which yields $\alpha x_{i+1} + x_{i+2} + x_{i+3} = \alpha + \beta$.

4.2 Empirical Interpolation of Degree 2 Polynomials

First let us mention that there exist 2^n distinct polynomials on a given set of n monomials. Since every query to the black-box returns a binary information (0 or 1), one cannot hope to exactly interpolate a polynomial in less than n queries.

Thus, when trying to interpolate a polynomial P known to be linear on 80 variables, the best method should require no more than 81 requests. This is easily achieved by first querying the polynomial on the 0 key (thus getting the coefficient of the 1 monomial) and then on the key whose only non-zero value is x_i, for $i \in \{1, \ldots, 80\}$. If $P(0, \ldots, 0) = P(0, \ldots, 0, x_i, 0, \ldots, 0)$ then x_i's coefficient is 0 in P and 1 otherwise.

Similarly, one can interpolate a polynomial of degree 2 in $\sum_{i=0}^{2} \binom{80}{i} = 3241$ requests, which is optimal. The process simply consists in first checking the presence of all linear monomials as described before and then checking for every pair $\{x_i, x_j\}$ if $P(\{x_i, x_j\}) + P(x_i) + P(x_j) = P(0)$. The process can also be done for a higher degree d, but the number of queries grows exponentially and quickly becomes impractical.

It has however been observed in every single cube attack result brought forth so far that the interpolated linear polynomials tend to be very sparse, most of them containing a single key bit and none of them more than 5. If we make the assumption that a linear black-box polynomial is actually a dictator (a single key bit), it can easily be found in about $1 + \lceil log_2(80) \rceil = 7$ queries. However this is not very accurate and because 81 queries is still reasonable, it is better to avoid doing this.

Yet this becomes incredibly useful when interpolating polynomials of degree 2. In theory, interpolating a polynomial of degree 2 over 80 variables would require $80 \times 81/2 = 3240$ cube computations. This can still be achieved in reasonable time for cubes of size 30 (obtained after 784 rounds), and allowed us to formally interpolate dozens of them but becomes unreasonable for polynomials of size 34 and more (obtained after 800 rounds).

However, every such polynomial interpolated so far had a very specific shape. It contains a single monomial of degree 2 which is of the form $x_i \cdot x_{i+1}$ accompanied by the monomial of degree 1 x_{i+2}, and possibly some other monomials of degree 1. The same result can be observed on the polynomials obtained in [11] after 704 rounds. This behavior can easily be explained by noticing that polynomials of this shape appear in the very first rounds of Trivium (they are part of the feedback function). It then does not come as a surprise that they are the most likely candidates to appear in the sparsest part of the polynomial representation of the cipher.

Now if we assume the quadratic polynomial detected has this shape, we can easily interpolate it. Indeed, it suffices to test the presence of every linear monomial (81 cubes need to be computed) and for every one detected, test the presence of the associated monomial of degree 2 as described earlier (most likely less than 10 extra cube computations). Then one can check whether the interpolated polynomial actually matches the behavior of the superpoly on the already computed cubes during the linearity and degree 2 tests. This works very well in practice and out of dozens of such polynomials interpolated after 799 rounds, this process only failed on one of them.

4.3 Results on 799 Rounds

Annex B lists all the linear and quadratic polynomials yielded after 799 rounds. The 6 quadratic polynomials, which were chronologically found first as subcubes of a large cube of size 40, have themselves a size between 34 and 37. The 12 linear polynomials, which were found later in a low-density large cube also of size 40, have a size between 32 and 33. As 12 key bits can be directly recovered thanks to the maxterms in less than 2^{37} queries, a full key recovery is possible by adding a phase of 2^{68} brute-force queries. Furthermore, due to the simplistic nature of the 6 quadratic polynomials, which can be evaluated in less than 2^{40} queries, the key space to be brute-forced can be reduced to 2^{62} elements. Indeed, by choosing an assignment for every variable appearing in a quadratic expression but a linear one, the value of this linear variable becomes immediately known. Because all of the 6 quadratic polynomials discovered have at least one such distinct linear variable, the complexity is reduced by a factor of 2^6. In this regard, up to a certain point, quadratic polynomials provide as much information as linear ones.

5 Conclusion

We have used the Moebius transform to analyze the off-line complexity of cube attacks on Trivium. We propose a new technique to decrease the density of the polynomials in Trivium and using this technique we were able to find 42 linear relations between key bits after 784 rounds for the first time, providing the first full key recovery attack at 784 rounds, feasible in about 15 min on a regular computer. Besides, we have recovered 12 key bits and 6 quadratic relations after 799 rounds of initialization of Trivium. We also provided a method to empirically select cubes yielding low degree polynomials.

A Key Bits Recovered After 784 Rounds

Cube indices	Superpoly
2,5,6,9,10,13,21,23,25,27,29,32,34,36,38,40,42,44,45,48,51,53,55,57,59,63,65,68,73,78	x_2
2,5,6,9,13,16,19,21,23,25,27,29,30,32,34,36,38,40,42,44,45,48,53,57,59,61,65,68,73,75,78	x_4
2,3,4,6,8,9,10,14,16,19,21,22,23,25,28,30,32,36,37,39,41,49,51,56,59,64,68,71,74,76,79	$x_7 + 1$
2,4,6,8,10,13,15,19,24,28,29,31,32,34,37,38,40,41,44,47,49,51,53,55,57,59,62,68,70,73,76,78	x_9
1,3,6,8,11,14,15,18,22,25,27,29,34,37,40,42,46,48,50,52,55,57,59,61,66,68,69,71,74,79	x_{11}
1,6,8,10,15,19,20,22,24,26,29,31,34,37,38,40,42,44,47,49,51,53,55,57,59,62,68,70,76,78	x_{19}
2,4,6,8,10,13,15,19,22,24,28,29,32,34,37,38,40,41,44,47,49,51,53,55,57,59,62,70,73,78	x_{20}
2,3,4,6,8,9,10,14,16,19,21,22,23,25,28,29,30,34,36,37,39,41,46,47,48,51,56,59,64,68,71,74,79	x_{21}
2,5,6,9,11,13,16,19,21,23,24,25,27,29,30,32,34,36,38,40,42,44,45,48,51,53,55,57,59,68,73	x_{22}
2,3,6,8,11,13,14,16,17,20,22,24,27,30,32,35,37,39,42,44,46,47,48,49,50,53,55,59,64,68,70,72,78	$x_{23} + x_{68}$

2,4,6,8,10,12,14,16,19,21,22,23,25,29,30,32,34,39,41,46,47,48,49,51,56,59,64,67,68,71,79	x_{24}
1,3,6,11,14,16,20,22,24,27,30,32,35,37,39,42,44,46,47,48,49,50,53,55,59,64,68,70,72,78	x_{25}
2,5,6,9,11,13,16,19,21,23,25,27,29,32,34,36,38,40,42,44,45,48,51,55,57,65,68,70,75,78	x_{26}
1,3,6,8,11,13,14,17,20,22,24,27,30,32,34,37,39,42,44,46,47,49,50,53,55,57,59,62,68,70,72,78	x_{35}
1,2,3,6,8,11,14,17,20,22,24,27,32,34,37,39,42,44,46,47,48,50,53,55,57,59,62,64,68,70,72	$x_{37} + 1$
2,3,6,8,11,13,14,16,20,22,24,27,30,32,34,35,37,39,42,44,46,48,50,53,55,57,59,62,68,72,78	x_{38}
2,4,6,8,10,12,14,16,21,23,25,30,32,34,36,37,39,41,46,47,48,49,51,56,59,64,67,68,71,79	$x_{39} + 1$
2,5,6,10,13,16,21,23,25,27,29,34,36,38,40,42,44,45,48,51,53,55,59,61,65,68,70,73,75,78	$x_{41} + 1$
2,4,6,8,10,12,13,15,19,20,22,24,26,31,34,37,38,42,44,47,49,53,55,57,59,68,70,73,76,78	x_{43}
4,6,8,10,13,15,19,20,24,26,28,31,34,37,38,40,41,42,44,47,49,51,53,55,57,59,68,70,73,76,78	x_{44}
1,4,6,8,10,13,15,19,20,22,24,26,28,29,32,34,38,40,41,42,49,51,53,55,57,59,62,68,70,76,78	x_{47}
2,4,6,8,10,13,15,19,20,22,24,26,28,29,31,34,37,38,40,42,44,53,55,57,59,62,68,70,76,78	x_{49}
2,4,6,8,13,15,19,20,22,24,26,28,32,34,37,38,40,42,47,49,51,53,55,57,59,62,70,73,76,78	x_{50}
2,5,9,10,13,16,21,23,25,27,29,30,32,34,36,38,40,42,44,45,48,53,55,57,59,63,65,68,75,78	x_{51}
2,4,6,8,10,13,15,19,20,24,26,31,34,37,38,40,42,44,47,49,51,53,55,57,59,68,70,73,76,78	x_{52}
2,5,6,9,13,16,23,25,27,29,30,34,36,38,40,42,44,45,48,51,53,55,57,59,61,63,65,68,70,78	x_{53}
1,4,6,8,10,12,13,15,19,20,22,24,26,28,31,34,38,40,41,42,44,47,49,55,57,59,68,70,73,76,78	x_{54}
2,4,6,8,10,13,15,19,22,24,26,31,32,37,38,40,42,44,47,49,51,53,55,57,59,68,70,73,76,78	x_{55}
2,4,6,8,10,13,15,20,22,24,26,28,31,34,37,38,40,42,44,47,49,51,53,55,57,59,62,68,70,73,78	x_{56}
2,4,6,8,10,13,15,19,20,22,24,26,28,29,31,32,34,37,38,40,42,44,47,51,53,57,59,62,70,73,76,78	x_{57}
1,4,6,8,10,13,15,20,22,24,26,28,31,32,34,37,38,41,42,47,49,53,55,57,59,68,70,73,76,78	x_{58}
2,4,6,8,10,12,14,16,19,21,22,23,25,28,29,30,34,36,37,39,41,46,48,51,56,59,64,71,76,79	x_{59}
1,4,6,8,10,13,15,19,20,22,24,26,28,29,31,34,38,40,42,44,49,51,53,55,57,59,68,70,76,78	x_{60}
1,4,6,8,10,12,13,15,19,22,24,26,29,31,34,37,38,42,44,47,49,51,53,55,57,59,68,70,73,78	x_{61}
2,4,6,8,15,19,20,22,24,26,29,31,32,34,37,38,40,42,44,47,49,51,53,55,57,59,68,70,76,78	x_{62}
2,4,6,8,13,15,19,20,22,24,26,28,31,32,34,40,41,42,47,49,51,53,55,57,59,68,70,73,76,78	x_{63}
2,4,6,8,10,13,15,19,20,22,24,26,28,29,31,32,34,40,42,44,47,49,53,55,57,59,68,70,76,78	x_{64}
2,4,6,8,10,12,15,19,20,22,24,26,28,29,32,34,37,40,42,44,47,51,53,55,57,59,62,68,70,78	x_{65}
2,4,6,8,10,13,15,19,20,22,24,26,28,29,32,34,37,42,44,47,49,51,53,55,57,59,68,70,76,78	x_{66}
2,3,4,6,8,10,14,16,19,21,22,23,25,28,30,32,34,36,37,39,41,46,51,56,59,64,68,71,74,76,79	x_{67}
2,6,8,10,13,15,19,20,22,24,26,28,29,31,32,34,37,38,40,41,42,44,51,53,55,57,59,68,70,76,78	$x_{68} + 1$
1,2,6,8,11,13,14,17,20,22,24,27,32,34,37,39,42,44,46,47,48,50,53,55,57,59,62,64,68,70,78	x_{69}

B TExpressions Recovered After 799 Rounds

Cube indices	Expression
0,2,4,5,6,7,9,11,13,14,15,18,20,22,24,26,32,35,37, 39,42,44,46,48,53,55,57,61,68,70,72,79	x_{25}
0,2,4,5,6,7,9,11,13,14,15,18,20,22,24,26,32,35,39, 42,44,46,48,52,55,57,62,68,70,74,76,79	$x_{25} + x_{40}$
0,2,4,5,6,7,9,11,13,14,15,18,20,22,24,26,32,35,37, 39,42,44,46,48,52,53,55,57,61,62,68,70,79	x_{36}
0,2,4,5,7,9,11,13,14,15,18,20,24,26,30,32,35,37,39, 40,42,44,46,48,52,53,55,62,68,70,74,79	x_{38}
0,4,5,6,7,9,11,13,14,15,18,20,22,24,26,30,32,35,37, 39,40,44,46,48,52,55,57,62,68,70,74,79	x_{42}
0,4,5,6,7,9,11,13,14,18,20,22,24,26,30,35,37,39,40, 44,46,48,52,55,57,62,68,70,72,74,76,79	x_{53}
0,2,4,5,6,9,11,13,14,15,17,18,19,20,22,24,26,30,32, 35,39,40,44,48,53,55,57,61,62,70,74,76,79	x_{58}

0,5,6,7,9,11,13,17,19,22,24,26,30,32,35,37,39,42, 44,46,48,52,53,55,57,61,62,68,72,74,76,79	x_{60}
0,2,4,5,6,7,9,11,13,15,17,18,19,22,24,26,30,32,37, 39,42,44,46,52,53,57,61,62,68,74,76,79	x_{62}
0,4,5,7,9,11,13,14,15,17,18,19,20,22,24,26,30,32, 35,37,39,40,44,48,53,55,61,68,72,74,76,79	x_{64}
0,4,5,6,7,9,11,13,15,18,20,22,24,26,30,32,35,37,39, 40,42,44,46,48,55,57,62,68,70,72,76,79	x_{66}
0,4,5,6,7,9,11,13,14,15,17,19,22,24,26,32,35,37,39, 40,42,46,48,52,55,57,62,68,70,74,76,79	x_{67}
0,2,4,6,8,11,13,16,19,21,23,26,28,30,32,34,36,38,40, 42,44,46,49,50,53,56,62,64,69,72,74,75,77,79	$x_9 + x_{34}x_{35} + x_{36}$
0,2,4,6,8,11,13,16,19,21,23,26,28,30,32,34,36,38,40, 42,44,46,50,53,56,58,62,64,69,72,74,75,77,79	$x_{22} + x_{47}x_{48} + x_{49}$
0,2,4,6,8,11,13,16,19,21,23,26,28,30,32,34,36,38,40, 42,44,46,49,50,53,56,58,59,62,69,71,74,75,79	$x_{24} + x_{49}x_{50} + x_{51}$
0,2,4,6,8,11,13,16,19,21,23,28,30,32,34,36,38,40,42, 44,46,49,50,53,56,59,62,64,66,69,72,74,75,77,79	$x_{11} + x_{36}x_{37} + x_{38}$
0,2,4,6,8,11,13,16,19,21,23,26,28,32,34,36,38,40,42, 44,46,50,53,56,58,59,62,64,66,69,71,72,74,75,77,79	$x_{52} + x_{77}x_{78} + x_{79}$
0,2,4,6,8,11,13,16,19,21,23,26,28,32,34,36,38,40,42, 44,46,48,50,52,53,56,58,59,62,66,69,71,72,74,75,77,79	$x_9 + x_{34}x_{35} + x_{36} + x_{61} + x_{17}x_{18} + x_{19}$

References

1. Alon, N., Kaufman, T., Krivelevich, M., Litsyn, S.N., Ron, D.: Testing low-degree polynomials over $GF(2)$. In: Arora, S., Jansen, K., Rolim, J.D.P., Sahai, A. (eds.) APPROX 2003 and RANDOM 2003. LNCS, vol. 2764, pp. 188–199. Springer, Heidelberg (2003)
2. Aumasson, J.-P., Dinur, I., Meier, W., Shamir, A.: Cube testers and key recovery attacks on reduced-round MD6 and trivium. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 1–22. Springer, Heidelberg (2009)
3. Blum, M., Luby, M., Rubinfeld, R.: Self-testing/correcting with applications to numerical problems. In: Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing, STOC '90, pp. 73–83. ACM, New York (1990)
4. De Cannière, C.: Trivium: a stream cipher construction inspired by block cipher design principles. In: Katsikas, S.K., López, J., Backes, M., Gritzalis, S., Preneel, B. (eds.) ISC 2006. LNCS, vol. 4176, pp. 171–186. Springer, Heidelberg (2006)
5. Chen, Z., Fu, B.: The complexity of testing monomials in multivariate polynomials. In: Wang, W., Zhu, X., Du, D.-Z. (eds.) COCOA 2011. LNCS, vol. 6831, pp. 1–15. Springer, Heidelberg (2011)
6. Crowley, P.: Trivium, sse2, corepy, and the gcube attackh (2008)
7. Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 278–299. Springer, Heidelberg (2009)
8. Dinur, I., Shamir, A.: Breaking grain-128 with dynamic cube attacks. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 167–187. Springer, Heidelberg (2011)
9. Joux, A.: Algorithmic Cryptanalysis, 1st edn. Chapman & Hall/CRC, Boca Raton (2009)

10. Knellwolf, S., Meier, W., Naya-Plasencia, M.: Conditional differential cryptanalysis of NLFSR-based cryptosystems. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 130–145. Springer, Heidelberg (2010)
11. Mroczkowski, P., Szmidt, J.: The cube attack on stream cipher trivium and quadraticity tests. Fundam. Inform. **114**(3–4), 309–318 (2012)
12. Vielhaber, M.: Breaking one.fivium by aida an algebraic iv differential attack. Cryptology ePrint Archive, Report 2007/413, (2007). http://eprint.iacr.org/