A Novel GPU-Based Implementation of the Cube Attack

Preliminary Results Against Trivium

Marco Cianfriglia[1,2(✉)], Stefano Guarino[2,3(✉)], Massimo Bernaschi[2], Flavio Lombardi[2], and Marco Pedicini[1,2]

[1] Roma Tre University, Rome, Italy
{cianfriglia,pedicini}@mat.uniroma3.it
[2] Istituto per le Applicazioni del Calcolo (IAC - CNR), Rome, Italy
{s.guarino,m.bernaschi,f.lombardi}@iac.cnr.it
[3] Sapienza University of Rome, Rome, Italy

Abstract. With black-box access to the cipher being its unique requirement, Dinur and Shamir's cube attack is a flexible cryptanalysis technique which can be applied to virtually any cipher. However, gaining a precise understanding of the characteristics that make a cipher vulnerable to the attack is still an open problem, and no implementation of the cube attack so far succeeded in breaking a real-world strong cipher. In this paper, we present a complete implementation of the cube attack on a GPU/CPU cluster able to improve state-of-the-art results against the Trivium cipher. In particular, our attack allows full key recovery up to 781 initialization rounds without brute-force, and yields the first ever maxterm after 800 initialization rounds. The proposed attack leverages a careful tuning of the available resources, based on an accurate analysis of the offline phase, that has been tailored to the characteristics of GPU computing. We discuss all design choices, detailing their respective advantages and drawbacks. Other than providing remarkable results, this paper shows how the cube attack can significantly benefit from accelerators like GPUs, paving the way for future work in the area.

Keywords: Cube attack · Trivium · GPU

1 Introduction

The security of a stream cipher relies on its ability to mimic the properties of the perfectly secure One Time Pad (OTP): predicting future keystream bits (*e.g.*, by recovering its inner state) must be computationally infeasible. In fact, as highlighted by *algebraic* and *correlation* attacks, any statistical correlation between output bits and linear combinations of input bits is a potential security breach for the cipher. Cryptographers are therefore caught in between implementation requirements, which suggest the use of efficient primitives such as *Feedback Shift Registers* (FSRs) or *Finite State Machines* (FSMs), and security requirements,

D. Gollmann et al. (Eds.): ACNS 2017, LNCS 10355, pp. 184–207, 2017.
DOI: 10.1007/978-3-319-61204-1_10

which demand for solutions able to disguise the dependence of keystream-bits on the inner state of the registers. Many recent stream ciphers therefore rely upon irregular clocks, mutual clock control, non-linear and/or mutual feedback among different registers, or combinations of these solutions.

The cube attack, proposed by Dinur and Shamir [10], can be classified as an algebraic known-plaintext attack. Assuming that a chunk of keystream can be recovered from a known plaintext-ciphertext pair, the attack allows determining a set of linear equations binding key-bits. However, cube attacks significantly deviate from traditional algebraic attacks in that the equations are not recovered symbolically, but rather extracted through exhaustive searches over selected public/IV bits – the edges of the *cubes* the attack is named after. The possibility that a cube yields a linear equation depends on both its size and on the algebraic properties of the cipher. Since the Algebraic Normal Form (ANF) of the cipher (that is, its representation as a binary polynomial) is generally unknown beforehand, in practice the attack usually runs without clear prior insights into a convenient strategy for selecting the cubes – an approach made possible by the fact that the attack only requires black-box access to the attacked cipher. Exploring cubes of different (possibly large) size, trying many different sets of indices, and varying the binary assignment of the public bits not belonging to the tested cube are all promising solutions, but they all come at an exponential cost. In a sense, cube attacks can be therefore assimilated to *Time-Memory-Data Trade-Off* (TMDTO) attacks, as their success rate strongly depends on the extensiveness of the pre-computation stage, on the memory available to store the results of that stage, and on the amount of data usable to implement it. Consequently, identifying the most favourable design choices is the main pillar of a possibly successful cube attack.

Contributions. The present paper motivates and discusses in depth an implementation for Graphics Processing Unit (GPU) of the cube attack. The target cipher is Trivium [8,22], already considered in the literature to test the viability of the cube attack [10,14]. Our contributions can be summarized as follows: (i) We tailor the design and implementation of the cube attack to the characteristics of GPUs, in order to fully exploit parallelization while coping with limited memory. Our framework is extremely flexible and can be adapted to any other cipher at no more cost than some fine (performance) tuning, mostly related to memory allocation. (ii) We show the performance gain with respect to a CPU implementation, including results obtained on latest generation GPU cards. (iii) Our implementation allows for exhaustively assigning values to (subsets of) public variables with negligible additional costs. This means extending the quest for superpolys to a dimension never explored in previous works, and, by not being tied to a very small set of IV combinations, potentially weakening one of the basic requirements of the cube attack, that is, the assumption of a completely tweakable IV. (iv) Even though we run the attack with only a few preliminary sets of cubes – specifically selected to both validate our code and compare our results with the literature – our findings improve on the state-of-the-art for attacks against reduced-round versions of Trivium.

Roadmap. This paper is organized as follows: Sect. 2 introduces the cube attack and the targeted cipher Trivium; our implementation of the attack is described in Sect. 3, whereas experimental results are reported and discussed in Sect. 4; Sect. 5 gives an overview of related works; finally, Sect. 6, draws conclusions and suggests possible directions for future work.

2 Preliminaries

In this section, we first describe the theoretical implant of the cube attack, and we then briefly introduce Trivium. More details about Trivium are reported in Appendix A.

The Cube Attack. Let z denote a generic keystream bit produced by a stream cipher $\mathcal{E}$. z is the result of a function $E : \mathbb{F}_2^{n+k} \to \mathbb{F}_2$, computed over the $n + k$ input bits obtained from an Initial Vector IV of length n and a secret key K of length k. It is well known that z can be expressed as $z = p(\mathbf{x}, \mathbf{y})$, where p is the polynomial representation of E, $\mathbf{x} = (x_1, \ldots, x_n)$ is the vector of public variables (IV), $\mathbf{y} = (y_1, \ldots, y_k)$ is the vector of secret variables (K), and all variables in p appear with degree 1, at most. The cube attack relies on extracting from p a set of linear equations binding private variables in $\mathbf{y}$, through a suitable offline pre-computation phase involving public variables in $\mathbf{x}$.

Let $I = \{i_1, \ldots, i_m\} \subset \{1, \ldots, n\}$ and let us introduce the complement $\overline{I} = \{1, \ldots, n\} \backslash I$ of the set I. With a slight abuse of notation, let us consider variables in $\mathbf{x}$ as partitioned by I: $\mathbf{x} = (\mathbf{x}_I, \mathbf{x}_{\overline{I}})$, *i.e.*, we tell apart the variables $\mathbf{x}_I$ indexed by I from those $\mathbf{x}_{\overline{I}}$ indexed by its complement $\overline{I}$. Let $t_I = x_{i_1} \cdots x_{i_m}$ be the monomial induced by I, that is, the product of all variables in $\mathbf{x}_I$. By writing $t_I(\mathbf{x}_I)$ we want to stress that t_I contains only variables in $\mathbf{x}_I$. If we factor $t_I(\mathbf{x}_I)$ out of $p(\mathbf{x}, \mathbf{y})$ we obtain

$$p(\mathbf{x}, \mathbf{y}) = t_I(\mathbf{x}_I) \cdot p_{S(I)}(\mathbf{x}, \mathbf{y}) + q(\mathbf{x}, \mathbf{y})$$

where the quotient $p_{S(I)}(\mathbf{x}, \mathbf{y})$ of the division is called the *superpoly* of I in p, whereas $q(\mathbf{x}, \mathbf{y})$ is the remainder of the division.

Now, for any binary vector $\mathbf{v}_{\overline{I}}$, we consider a fixed assignment for variables $\mathbf{x}_{\overline{I}}$[1], and let $C_I(\mathbf{v}_{\overline{I}})$ denote the *cube* induced by I and $\mathbf{v}_{\overline{I}}$, that is, the set of all 2^m possible binary assignments to $\mathbf{x}$ in which variables $\mathbf{x}_{\overline{I}}$ assume values specified by the binary vector $\mathbf{v}_{\overline{I}}$ and the remaining variables in $\mathbf{x}_I$ take all the possible combinations. It is easy to verify that all monomials in $p_{S(I)}$ do not contain any of the variables $\mathbf{x}_I$ (*i.e.*, $p_{S(I)}(\mathbf{x}, \mathbf{y}) = p_{S(I)}(\mathbf{x}_{\overline{I}}, \mathbf{y})$), whereas all monomials in q do not contain *at least* one of the variables in $\mathbf{x}_I$. For this reason, regardless of $\mathbf{y}$, the sum of $p(\mathbf{x}, \mathbf{y})$ over all elements $\mathbf{v}$ of $C_I(\mathbf{v}_{\overline{I}})$ yields [10]

$$\sum_{\mathbf{v} \in C_I(\mathbf{v}_{\overline{I}})} p(\mathbf{v}, \mathbf{y}) = p_{S(I)}(\mathbf{v}_{\overline{I}}, \mathbf{y}) \tag{1}$$

which obviously does not depend on variables $\mathbf{x}_I$ anymore.

[1] The standard assumption is $\mathbf{v}_{\overline{I}} = \mathbf{0}$, but this is not actually required.

If $p_{S(I)}(\mathbf{v}_{\overline{I}}, \mathbf{y})$ is *linear*, the monomial $t_I(\mathbf{x}_I)$ is called a *maxterm for p with the assignment* $\mathbf{v}_{\overline{I}}$. If we can identify maxterms and find the symbolic expression of their superpolys, we obtain a system of linear equations that can be used to recover the secret key.

As $\mathbf{v}_{\overline{I}}$ is always clear from the context, to improve readability in the following we simply denote $C_I(\mathbf{v}_{\overline{I}})$ and $p_{S(I)}(\mathbf{v}_{\overline{I}}, \mathbf{y})$ as C_I and $p_{S(I)}(\mathbf{y})$, respectively.

Trivium. Trivium [8] is a stream cipher conceived by Christophe De Cannière and Bart Preneel, part of the eSTREAM portfolio. It generates up to 2^{64} bits of output from an 80-bit key K and an 80-bit Initial Vector IV, and it shows remarkable resistance to cryptanalysis despite its simplicity and its excellent performance. Trivium is composed by a 288-bit internal state consisting of three shift registers of length 93, 84 and 111, respectively. The feedback to each of these registers and the output bit of the cipher are obtained through non-linear combinations involving in total 15 out of the 288 internal state bits. To initialize the cipher, K and IV are written into two of the shift registers, with a fixed pattern filling the remaining bits. 1152 initialization rounds guarantee that the output begins to be produced only after all key-bits and IV-bits have been sufficiently mixed together to define the internal state of the registers.

3 The Proposed GPU Implementation of the Attack

In this section, we present, detail, and discuss our attack, designed to run on a cluster equipped with Graphics Processing Units (GPU). As previously mentioned, the success of a cube attack is highly dependent on suitable implementation choices. In order to better explain our own approach, we start with an analysis of the cube attack from a more implementative perspective.

3.1 Practical Cube Attack

At a high level, any practical implementation of the cube attack requires performing the following steps:

S1 Find as many maxterms as possible;
S2 For each maxterm, find the corresponding linear equation(s);
S3 Solve the obtained linear system.

Step S1. This is the core of the attack, where cubes that yield linear equations are identified. Choosing candidate maxterms (*i.e.*, cubes) is non-trivial. Intuitively, the degree of most maxterms lies in a specific range that depends on the (unknown) degree distribution of the monomials of the polynomial p. If the degree of t_I is too small, then $p_{S(I)}$ is most likely non-linear, but if the degree of t_I is too large, then $p_{S(I)}$ will probably be constant (*e.g.*, null). Moreover, since

the complexity of the offline phase scales exponentially with $|I|$, the degree of tested potential maxterms is strongly influenced by practical limitations.

In [10], the authors propose a *random walk* to explore a maximal cube $C_{I_{\max}}$, *i.e.*, starting from a random subset $I \subset I_{\max}$ and iteratively testing the superpoly $p_{S(I)}$ to decide whether the degree of t_I should be increased or decreased. The underlying idea is to use a probabilistic approach to identify the optimal size $|I|$. In [14], the authors evaluate the cipher upon all vertices of a maximal cube $C_{I_{\max}}$, store the results in a table T of size $|T| = 2^{|I_{\max}|}$, and then apply the *Moebius transform* to the entire table T, thus computing at once the sums over $\binom{|I_{\max}|}{d}$ sub-cubes of $C_{I_{\max}}$ of degree d, for $d = 0, \ldots, |I_{\max}|$. These cubes are all possible sub-cubes of $C_{I_{\max}}$ in which the variables outside the cube have been set equal to 0. In this case the rationale is minimizing processing cost by reusing partial computations as much as possible. Interestingly, the authors of [14] show that specific cubes perform better than others, at least for reduced-round variants of Trivium, and use their findings to select the most promising maximal set $I_{\max}$.

None of these two strategies is suitable for GPUs. The stochastic nature of the random walk prevents the sequence of steps from being determined *a priori*, since the computation is performed only when (and if) needed. On the other hand, the Moebius transform requires a rigid schema of calculations and a large number of alternating read and write operations in memory that must be synchronized. Both approaches are conceived for implementations in which computational power is a constraint (while memory is not), and all advantages of using the Moebius Transform are lost in case of parallel processing. We rather perform an exhaustive search over a portion of a maximal cube, a solution that is highly parallelizable and feasible with our computational resources.

For each candidate maxterm t_I, we need to verify whether the superpoly $p_{S(I)}$ is linear. The goal being recovering key bits, any fixed assignment of variables $\mathbf{x}_{\overline{I}}$ with the bit vector $\mathbf{v}_{\overline{I}}$ can be used to get rid of variables. In order to guarantee that the degree of each superpoly is reduced to the bare minimum, the assignment $\mathbf{v}_{\overline{I}}$ to $\mathbf{0}$ is usually preferred, but we argue that this is not necessarily the best choice, as motivated later in Sect. 4.2. In any case, at this stage the superpoly $p_{S(I)}$ only depends on $\mathbf{y}$. In principle, assessing the linearity of $p_{S(I)}(\mathbf{y})$ requires finding all of its coefficients, but efficient *probabilistic* linearity tests [7,21] can safely replace deterministic ones in most practical settings. Probabilistic tests involve verifying if

$$p_{S(I)}(\mathbf{u}_1 + \mathbf{u}_2) = p_{S(I)}(\mathbf{u}_1) + p_{S(I)}(\mathbf{u}_2) + p_{S(I)}(\mathbf{0}) \tag{2}$$

holds for random pairs of vectors $\mathbf{u}_1, \mathbf{u}_2$. Practically, this means evaluating *numerically* four sums: $\sum_{\mathbf{v} \in C_I} E(\mathbf{v}, \mathbf{0})$, $\sum_{\mathbf{v} \in C_I} E(\mathbf{v}, \mathbf{u}_1)$, $\sum_{\mathbf{v} \in C_I} E(\mathbf{v}, \mathbf{u}_2)$, and $\sum_{\mathbf{v} \in C_I} E(\mathbf{v}, \mathbf{u}_1 + \mathbf{u}_2)$.

Probabilistic tests rely on the fact that (2) must be true for all $\mathbf{u}_1, \mathbf{u}_2$ if $p_{S(I)}$ is linear, whereas, in general, it holds with probability $\frac{1}{2}$. In particular, as done for previous cube attacks [10,14], we will resort to a *complete-graph test* [21], which guarantees a slightly lesser accuracy than the (truly-random) BLR test [7] with far fewer evaluations of $p_{S(I)}$. Let us remark that what ultimately matters in

the envisaged scenario is identifying "far-from-linear" superpolys [20]. To clarify, let us consider the superpoly $p_{S(I)}$

$$p_{S(I)}(\mathbf{y}) = l(\mathbf{y}) + \prod_{i=1}^{k} y_i$$

formed by a sum $l(\mathbf{y})$ of linear terms, plus one nonlinear term given by the product of all variables in $\mathbf{y}$. Despite the equality $p_{S(I)}(\mathbf{y}) = l(\mathbf{y})$ is *formally* wrong (the degree of $p_{S(I)}$ is as large as k), $p_{S(I)}(\mathbf{u}) = l(\mathbf{u})$ is *numerically* correct for all $\mathbf{u} \in \mathbb{F}_2^k$, except $\mathbf{u} = (1, 1, \ldots, 1)$. In other words, mistaking $p_{S(I)}$ for linear has practical consequences only if $\mathbf{u} = (1, 1, \ldots, 1)$.

Steps S2 and S3. Step **S2** consists in finding the *symbolic* expression of the superpoly of all identified maxterms, and the free term of the corresponding equation. Again, this turns into a set of *numerical* evaluations: the free term of $p_{S(I)}(\mathbf{y})$ is

$$p_{S(I)}(\mathbf{0}) = \sum_{\mathbf{v} \in C_I} E(\mathbf{v}, \mathbf{0})$$

whereas the coefficient of each variable y_i is

$$p_{S(I)}(\mathbf{e}_i) + p_{S(I)}(\mathbf{0}) = \sum_{\mathbf{v} \in C_I} E(\mathbf{v}, \mathbf{e}_i) + \sum_{\mathbf{v} \in C_I} E(\mathbf{v}, \mathbf{0})$$

where $\mathbf{e}_i$ is the unit vector with all null coordinates except $y_i = 1$. Once the polynomial $p_{S(I)}(\mathbf{y})$ is found, the attack assumes the availability of the 2^m keystream bits produced in correspondence to a fixed (unknown) assignment to the variables $\mathbf{y}$, as the variables $\mathbf{x}$ take all possible assignments in C_I. This produces the linear equation

$$p_{S(I)}(\mathbf{y}) = \sum_{\mathbf{v} \in C_I} E(\mathbf{v}, \mathbf{y})$$

whose left side is a linear combination of the key variables $\mathbf{y}$ with coefficients found *offline*, whereas the right side is a number found *online*, and whose solution is the sought unknown assignment to $\mathbf{y}$.

Finally, Step **S3** just requires solving the obtained linear system with any suitable technique described in the literature.

3.2 The Setting

Generally speaking, GPUs are processing units characterized by the following advantages and limitations:

Computing: Each unit features a large number (*i.e., thousands*) of simple cores, that make possible running a much higher number of parallel threads compared to a standard CPU. More precisely, the GPU's basic processing unit is the *warp* consisting of 32 threads each. Threads are designed to work on 32-bit words, and the performance is maximized if all threads belonging to the same warp execute exactly the same operations at the same time on different but contiguous data.

Memory: The so-called *global* memory available on a GPU is limited, typically between 4 and 12 GB. Each thread can independently access data (random access is fully supported, but costly performance-wise). However, when threads in a warp access consecutive 32-bit words, the cost is equivalent to a single memory operation. Concurrent readings and writings by different threads to the same resources, which require some level of synchronization, should be avoided to prevent serialization that defeats parallelism.

The basic step of the attack is the sum of $E(\mathbf{v}, \mathbf{y})$ over all elements $\mathbf{v}$ of a cube C_I. Each time we sum over a cube, the key variables $\mathbf{y}$ are fixed, either to a random $\mathbf{u}_j$ for the linearity tests, or to $\mathbf{0}$ and to versors $\mathbf{e}_i$ for determining the superpoly. In both cases exactly the same sum $\sum_{\mathbf{v} \in C_I} E(\mathbf{v}, \mathbf{u}_j)$ must be performed for all elements of a set of keys $\{\mathbf{u}_1, \ldots, \mathbf{u}_M\}$.

We define the following strategy for carrying out the sums over a cube with the goal of maximizing the parallelization and fully exploiting at its best the computational power offered by GPUs:

- Assigning to all the threads within a warp the computation of the same cube C_I but with a different key $\mathbf{u}_j$. This choice guarantees that all threads perform the same operation at the same time for the entire computation.
- Leveraging the GPU computational power to calculate all the elements of a cube C_I, providing to the threads just a bit-mask representing the set I. With this approach we can exploit all available GPU memory to store the cubes evaluations and minimize, at the same time, the number of memory access operations.
- Defining a keystream generator function $E(\mathbf{x}, \mathbf{y})$ which outputs a 32-bit word, and letting each thread work on the whole word, fully leveraging the GPU computing model. This approach offers two remarkable benefits: (i) considering 32 keystream bits altogether is equivalent to concurrently attacking 32 different polynomials, and (ii) working on 32-bit integers fits much better with the GPUs features, whereas forcing the threads to work on single bits would critically affect the performance of the attack. Therefore, attacking 32 keystream bits altogether reduces (of a factor 32) the memory needed for storing the cubes' evaluation, thus imposing some limitations on the size of the cubes to be tested, as we will clarify later.
- Choosing the number M of keys to be a multiple of the warp size in order to perform the probabilistic linearity test on 32 keystream bits at the same time and for all M keys.

3.3 The Attack

A severe constraint in any GPU implementation is represented by the amount of memory $|T|$ currently available on GPUs. Moreover, for each cube, we need to consider M different keys in order to run the linearity test, thus reducing the amount of available memory even further to $|T|/M$. Storing single evaluations of the cipher in T means testing only sub-cubes of a maximal cube of size $|I_{\max}| = \log_2(|T|/M)$. With the memory available in current GPUs, $\log_2(|T|/M)$ is not large enough for any reasonably strong cipher. The new approach we propose is highly parallelizable, it can fully exploit the computational resources offered by GPU, and it is able to exploit GPU memory to test high order maximal cubes.

The proposed design of the attack relies on the following rationale: exploring only a portion of the maximal cube $C_{I_{\max}}$, considering only subsets $I \subseteq I_{\max}$ characterized by a non-empty *minimal* intersection $I_{\min}$. Quite naturally, a similar design leads to two distinct CUDA[2] kernels, respectively responsible for: (1) computing many variants of the cube $C_{I_{\min}}$, one for each of the possible combinations of the indices in $I_{\max} \setminus I_{\min}$, and writing the results in memory; (2) combining the stored results to test all cubes C_I such that $I_{\min} \subseteq I \subseteq I_{\max}$. Following this approach, the size of the explored $I_{\max}$ can be raised to $|I_{\max}| = |I_{\min}| + \log_2(|T|/M)$, with read and write memory operations carried out by different kernels.

According to the notation introduced in Sect. 2, the public variables are $\mathbf{x} = (x_1, \ldots, x_n)$. Now, let us distinguish these n public variables into three sets $\mathbf{x}_{\text{fix}} = (x_{i_1}, \ldots, x_{i_{d_{\text{fix}}}})$, $\mathbf{x}_{\text{free}} = (x_{j_1}, \ldots, x_{j_{d_{\text{free}}}})$, and $\mathbf{x}^*$, of size d_{fix}, d_{free}, and $n - d$, respectively, where $d = d_{\text{fix}} - d_{\text{free}}$. The variables $\mathbf{x}_{\text{fix}}$ correspond to the *fixed* components of $C_{I_{\max}}$ identified by $I_{\min}$, *i.e.*, $I_{\min} = \{i_1, \ldots, i_{d_{\text{fix}}}\}$, whereas the variables $\mathbf{x}_{\text{free}}$ correspond to the remaining *free* components of $C_{I_{\max}}$, *i.e.*, $I_{\max} \setminus I_{\min} = \{j_1, \ldots, j_{d_{\text{free}}}\}$ and $|I_{\max}| = d$. The variables $\mathbf{u}^*$ are the remaining public variables that fall outside $I_{\max}$.

The two kernels of our attack can be described as follows:

Kernel 1: It uses $2^{d_{\text{free}}}$ warps. Since, as described before, the 32 threads belonging to the same warp perform exactly the same operations but for different keys, in the following we simply consider a representative thread per warp and ignore the private variables $\mathbf{y}$.[3] For $t = 0, \ldots, 2^{d_{\text{free}}} - 1$, thread (*i.e.*, warp) s sums $E(\mathbf{u}, \mathbf{y})$ over each vertex of the cube $C^s_{I_{\min}}$ of size d_{fix} determined by the assignment of the d_{free}-bit representation $\mathbf{u}_{\text{free}}$ of integer s to the variables $\mathbf{x}_{\text{free}}$ and of $\mathbf{0}$ to the variable $\mathbf{u}^*$. Finally, thread s writes the sum in the s^{th} entry of table T, so that, at the end of the execution of the kernel, each entry of T contains the sum over a cube of size d_{fix}. These evaluations allow for testing the monomial $t_{I_{\min}}$ with all the aforementioned assignments to the other $n - d_{\text{fix}}$ variables.

[2] CUDA is the software framework used for programming Nvidia GPUs.

[3] The work that here is assigned to a single thread can be actually split among any number of threads, reassembling the results at the end. We will not consider this possibility here for the sake of clarity.

Kernel 2: By simply combining the values stored in T at the end of Kernel 1, it is now possible to explore cubes of potentially any size $d_{\text{fix}} + \delta$, with $0 \leq \delta \leq d_{\text{free}}$. Although the exploration can potentially follow many other approaches (*e.g.*, a random walk as in [10]), the large computing power of our platform suggests to test cubes exhaustively. Moreover, we extend the exhaustive search to an area never reached, to the best of our knowledge, in the literature. For all I such that $I_{\min} \subseteq I \subseteq I_{\max}$, this kernel considers all variants of cube C_I obtained assigning all possible combinations of values to the variables in $I_{\max} \setminus I$. More precisely, for each possible choice of $\delta \in [0, d_{\text{free}}]$, there are exactly $\binom{d_{\text{free}}}{\delta} 2^{d_{\text{free}}-\delta}$ distinct cubes of size $d_{\text{fix}} + \delta$ available. In fact, we can choose δ free variables (the additional dimensions of the cube) in $\binom{d_{\text{free}}}{\delta}$ different ways, and we can choose the fixed assignment to the remaining $d_{\text{free}} - \delta$ variables in any of the $2^{d_{\text{free}}-\delta}$ possible combinations.

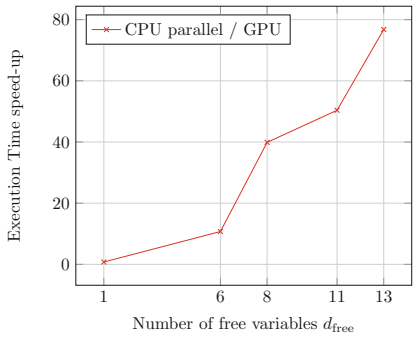
As a matter of fact, the number of cubes considered in [14] is $\sum_{\delta=0}^{d_{\text{free}}} \binom{d_{\text{free}}}{\delta} = 2^{d_{\text{free}}}$, whereas the number of cubes tested by our approach is significantly larger, namely, $\sum_{\delta=0}^{d_{\text{free}}} 2^{d_{\text{free}}-\delta} \binom{d_{\text{free}}}{\delta} = 3^{d_{\text{free}}}$. We would like to highlight that Kernel 2 is computationally dominated by Kernel 1, so the cost of our exhaustive search is negligible. Therefore, our design entails considering any possible assignment to variables outside the cube, to finally address the common conjecture (never proved in the literature), that assigning **0** is the best possible solution.

Let us underline that, in order to validate our implementation of the cube attack, we symbolically evaluated the polynomial p of Trivium up to 400 initialization rounds, and used p to identify all possible maxterms and their superpoly. We then ran the attack to find all maxterms whose variables belonged to selected sets I. Our experimental findings matched the symbolical findings. Further experimental validation of our code is reported in Sect. 4.

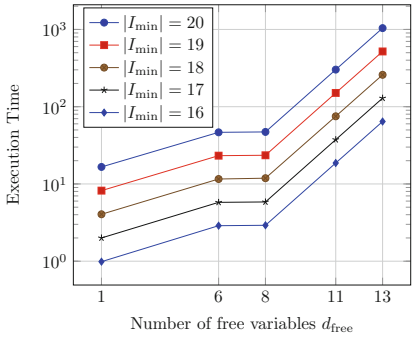
3.4 Performance Analysis

To evaluate the performance of our GPU based solution, we developed both a CPU and a GPU version of the cube attack. The cluster we used for the experiments is composed by 3 nodes, each equipped with 4 Tesla K80 with 12 GB of *global* memory and 4 Intel Xeon CPU E5-2640 with 128 GB of RAM. The CPU experiments were conducted on a parallel version based on OpenMP that exploits 32 cores of the four Intel(R) Xeon(R) CPU E5-2640. Each performance test was executed 5 times and the average time is reported. It is worth noticing that all versions rely on the same base functions to implement Trivium.

In Fig. 1a, we report the speed-up gained by the GPU version with respect to the parallel CPU version. We evaluated the two solutions over growing size maximal cubes $C_{I_{\max}}$, in which we anchor the size of $I_{\min}$, consequently causing the size of the set $I_{\max} \setminus I_{\min}$ to exponentially increase. Overall, the experiments show that the benefit of using the GPU version grows with the number of free variables d_{free} considered, reaching a speed-up up to 70× when $d_{\text{free}} = 13$. The rationale is that the execution time of the CPU version increases almost linearly with d_{free} from the very beginning, whereas a similar trend can be observed for



(a) Speedup of parallel CPU vs. GPU



(b) GPU performance analysis

Fig. 1. Performance experiments

the GPU version only when the number of blocks in use gets larger than the number of Streaming Multiprocessors (SMs) of the GPU, which happens when $d_{\text{free}} \geq 9$ in our case. Of course, slight fluctuations are possible, mostly due to the complex interactions among the multiple cache levels of a modern CPU. Moreover, we evaluated how the GPU solution scales when d_{free} increases. As reported in Fig. 1b, our solution scales linearly with the size of the problem, *i.e.*, exponentially with the size of the sub-cubes $C_{I_{\min}}$, thus paving the way for future works in the area.

Finally, we ran the attack under the control of the Nvidia profiler in order to measure the ALU occupancy achieved by our kernels. Kernel 1 is invoked just once per run to fill the whole table T, with an occupancy consistently over 95% when $d_{\text{free}} \geq 10$. Kernel 2 is instead invoked once per each $\delta \in [0, d_{\text{free}}]$, to compute all available cubes of size $d_{\text{fix}} + \delta$. The maximum occupancy exceeds 95% as soon as $d_{\text{free}} \geq 12$, with an average of approximately 50%. In either case the impact of d_{fix}, which determines the load of each thread, is negligible. Considering that d_{free} should be maximized to improve the attack success rate, our kernels guarantee an excellent use of resources in any realistic application. For instance, in our experiments discussed in Sect. 4 we set $d_{\text{free}} = 16$, which guarantees an occupancy above 99% for Kernel 1, and a maximum occupancy above 98% for Kernel 2.

4 Results

Finally, this section reports the results obtained by our GPU implementation of the cube attack against reduced-round Trivium. We recall that the attack ran on a cluster composed by 3 nodes, each equipped with 4 Tesla K80 with 12 GB of *global* memory and 4 Intel Xeon CPU E5-2640 with 128 GB of RAM.

As mentioned in Sect. 3.3, we performed a formal evaluation of our implementation, by checking our experimental results against Trivium's polynomials, explicitly computed up to 400 initialization rounds. In the following, the number

of initialization rounds instead matches (and slightly overtakes) the best results from the literature, thus reaching a point where a symbolic evaluation would be prohibitive. Still, the results we exhibit are obtained from experiments specifically designed to reproduce tests carried out in the recent past [14], so as to provide, at the same time: (i) a direct comparison of our results with the state-of-the-art; (ii) an immediate means to assess the advantages of our approach, and (iii) a further validation of the correctness of our code.

Experimental Setting. In our attack, we consider two different reduced-round variants of Trivium, corresponding to 768 and 800 initialization rounds, respectively. As explained and motivated in Sect. 3.2, in our scheme, each call to Trivium produces 32 key-stream bits, which we use in our concurrent search for superpolys. The most significant practical consequence of a similar construction is the ability to devise attacks to Trivium reduced to any number of initialization rounds ranging from 768 to 831, at the cost of just two attacks, although the number of available superpolys decreases with the number of rounds. As a matter of fact, the j^{th} output bit after 768 rounds can also be interpreted as the $(j-i)^{th}$ bit of output after $768+i$ initialization rounds, for any $j \geq i$. In other words, an attack to Trivium reduced to $768+i$ initialization rounds can count upon all superpolys found in correspondence of the j^{th} output bit after 768 rounds, for all $j \geq i$.

For each of the two attacks (768 and 800 initialization rounds), we ran a set of independent runs, each using a different choice for the pair of sets of variables $I_{\min}, I_{\max}$ (with $I_{\min} \subset I_{\max}$) that define the minimal and maximal tested cubes $C_{I_{\min}}$ and $C_{I_{\max}}$. The size of $I_{\min}$ and $I_{\max} \setminus I_{\min}$ is $d_{\text{fix}} = 25$ and $d_{\text{free}} = 16$, respectively, for all runs, so that all maximal cubes have size $d = d_{\text{fix}} + d_{\text{free}} = 41$. Peculiarly to our implementation, when we test the monomial composed of all variables in some set $I_{\min} \subseteq I \subseteq I_{\max}$, we exhaustively assign values to all public variables in $I_{\max} \setminus I$, thus concurrently testing the linearity of $2^{41-|I|}$ possibly different superpolys. This feature of our attack – a possibility overlooked in the literature, but almost free-of-charge in our framework – provides primary benefits, as described in Sect. 4.2.

In all the reported experiments, we use a complete-graph linearity test based on combining 10 randomly sampled keys.

4.1 Summary of Results

As mentioned before, we implemented two attacks, against Trivium reduced to 768 (Trivium-768 in the following) and 800 (Trivium-800) initialization rounds, respectively. In both cases, our setting allows obtaining superpolys corresponding to 32 output bits altogether, at the cost of a single attack.

Results Against Trivium-768. For the attack against Trivium-768, we took inspiration from [14]: we launched 12 runs based on 12 different pairs $I_{\min}, I_{\max}$, chosen so as to guarantee that each of the 12 linearly independent superpolys

found in [14] after 799 initialization rounds was to be found by one of our runs. The rationale of reproducing results from [14] was to both test the correctness of our implementation, and provide a better understanding of the advantages of our implementation with respect to the state-of-the-art. In this sense, let us highlight that a single run of ours cannot be directly compared with all results presented in [14], because each of our runs only explores the limited portion of the maximal cube $C_{I_{\max}}$ composed by all super-cubes of $C_{I_{\min}}$.

To better describe our results, let us introduce the binary matrix A whose element $A(i, j)$ is the coefficient of variable y_j in the i^{th} available superpoly. The rank of A, denoted $\mathrm{rk}(A)$, clearly determines the number of key bits that can be recovered in the online phase of the attack based on the available superpolys, before recurring to brute-force.

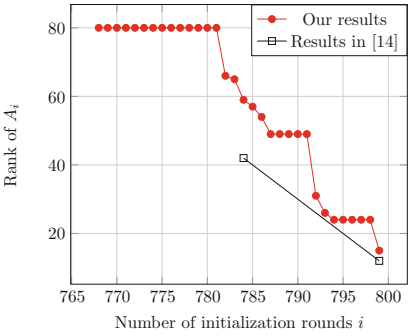
As described before, the superpolys yielded by the i^{th} output bit after round 768 are usable to attack Trivium for any number of initialization rounds between 768 and $768+i$. It is possible to define 32 different matrices $A_{768}, \ldots, A_{799}$: A_{768} includes all superpolys found, while each matrix A_{768+i} is obtained by incrementally removing the superpolys yielded by output bits $0, \ldots, i-1$. Figure 2a shows $\mathrm{rk}(A_i)$ as a function of i, comparing our findings with those of [14].

Overall, our results extend the state-of-the-art in a remarkable way, especially if we consider that our quest for maxterms was circumscribed to multiples of 12 *base* monomials of degree 25. In particular, let us highlight a few aspects that emerge from Fig. 2a:

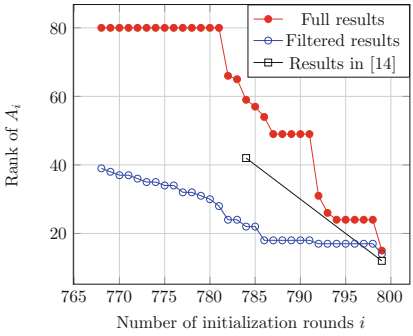
- Since our runs were designed to include all 12 maxterms found in [14] after 799 initialization rounds, it is not surprising that $\mathrm{rk}(A_{799})$ is at least 12. Yet, it is indeed larger: we found 3 more linearly independent superpolys, reaching $\mathrm{rk}(A_{799}) = 15$.
- Although we did not force our tested cube to include the maxterms found in [14] after 784 rounds, we have $\mathrm{rk}(A_{784}) = 59$, compared with rank 42 found in [14].
- Finally, and probably most importantly, our attack allows a full key recovery up to 781 initialization rounds.

Selected superpolys that guarantee the above ranks are reported in Appendix B, together with the corresponding maxterms. Very interesting is also *how* novel superpolys were found, a point that is better described in the following.

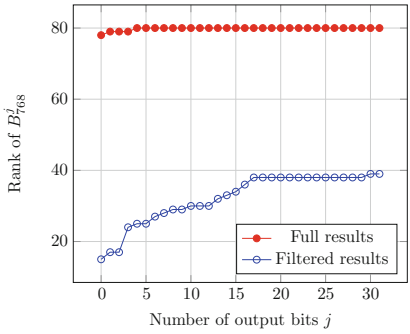
Results Against Trivium-800. To provide a further test of the quality of our attack, we launched a preliminary attack against Trivium-800. We kept unvaried all the parameters of the attack ($d_{\mathrm{fix}} = 25$, $d_{\mathrm{free}} = 16$, 32 output bits attacked altogether), but this time we only launched 4 runs, and we chose the sets $I_{\min}, I_{\max}$ at random. In total, we were able to find a single maxterm corresponding to 800 rounds, and no maxterms afterwards. This maxterm and the corresponding superpoly are also reported in Appendix B. Although our findings only allow to cut in half the complexity of a brute force attack, this is the first ever superpoly found considering more than 799 initialization rounds. We



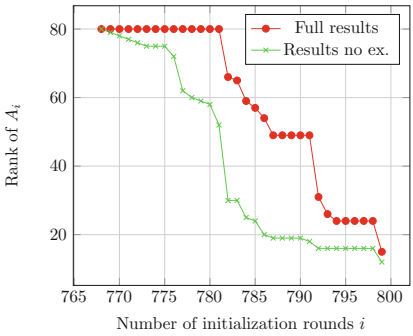
(a) Comparison with previous work



(b) Impact of probabilistic linearity



(c) Impact of using 32 output bits



(d) Impact of exhaustive search

Fig. 2. Our results

recall that our limited results should not appear as surprising: as previous work suggests [10,14], when the number of initialization rounds grows, a cube attack should increase the average degree of candidate maxterms and/or implement specific strategies for the selection of the index sets [14].

4.2 Further Discussion

Hereafter, we provide a more detailed analysis and a further discussion of our findings, considering two aspects in particular: the reliability of commonly used linearity tests, and the peculiar advantages of our attack design. Unless otherwise specified, in the following we always focus on Trivium-768.

On Probabilistic Linearity. A common practice in the cube attack related literature consists in using a probabilistic linearity test, meaning that a (small) chance exists that the superpolys found by an attack are not actually linear. In particular, the best results obtained with the cube attack against Trivium use a complete-graph test, which, with respect to the standard BLR test, trades-off

accuracy for efficiency. The viability of a similar choice is supported by previous work [12,21], showing that the complete-graph test behaves essentially as a BLR test in testing a randomly chosen function f, with the quality of the former being especially high if the nonlinearity (minimum distance from any affine function) of f is large, that is, when the result of the test is particularly relevant.

Following the trend, we chose to implement a complete-graph test based on a set of 10 randomly chosen keys, exactly as done in [14]. However, while increasing the number of tests done *during* the attack was costly for us (it impacts on memory usage), implementing further test on the superpolys found *at the end* of the attack was not. We therefore decided to put our superpolys through additional tests involving other 15 keys chosen uniformly at random. Figure 2b compares $\mathrm{rk}(A_i)$ as a function of i, for our full results and our filtered results, in which all superpolys that failed at least one of the additional tests have been removed. Let us stress once more that these two sets of results cannot be defined as wrong and correct, but they rather correspond to two different levels of trust in the found superpolys. In a sense, choosing between the two sets is equivalent to selecting the desired trade-off between efficiency and reliability of the attack: our full results permit a faster attack, which however may fail for a subset of all possible keys. Of course, many middle ways/intermediate approaches are possible. Investigating whether the reason of these failing tests is related to any of our design choices is left to future work.

On Using 32 Output Bits. A significant novelty of our implementation consists in the ability to concurrently attack 32 different polynomials, which describe 32 consecutive output bits of the target cipher. This choice is induced by GPUs features – as discussed in Sect. 3.2 – yet it is natural to assess what benefits it introduces. In Sect. 4.1 we showed that looking at 32 output bits altogether can be considered a way to concurrently attack 32 different reduced-round variants of Trivium. However, aiming to extend the attack to the full version of the cipher, our implementation can be used to check whether the same set of monomials yield different superpolys, hopefully involving different key variables, when we focus on different output bits. To this end, let us introduce a new set of matrices $B^0_{768}, \ldots, B^{31}_{768}$, where each B^j_{768} is obtained considering only the superpolys yielded by output bits $0, \ldots, j$ after 768 initialization rounds (*i.e.*, $A_{768} = B^{31}_{768}$). Figure 2c shows $\mathrm{rk}(B^j_{768})$ as a function of j, for both our full results and our filtered results. What the figure highlights is that considering several output bits altogether for the same version of the cipher, albeit possibly causing issues related to memory usage, does introduce the expected benefit, indeed a remarkable benefit if the matrix rank is initially (*i.e.*, when $j = 0$) low. This is the first ever result showing that considering a larger set of output bits is a viable alternative to exploring a larger cube.

On the Advantages of the Exhaustive Search. As described before, our implementation allows to find significantly more linearly independent superpolys than previous attempts from the literature. One of the reasons of our findings

is the parallelization that makes possible to carry out, at a negligible cost, an exhaustive search over all public variables in $I_{\max} \setminus I$ when the cube C_I is under test. Figure 2d, again focusing on $\mathrm{rk}(A_i)$, compares our full results with results obtained without exhaustive search (shortened "no ex."), *i.e.*, setting all variables in $I_{\max} \setminus I$ to 0, as usually done in related work. What emerges is that through an exhaustive search it is indeed possible to remarkably increase $\mathrm{rk}(A_i)$. Significantly, the exhaustive search is what allows us to improve on the state-of-the-art for $i = 799$, which, among other things, suggests that the benefits of the exhaustive search are particularly relevant when increasing the number of tested cubes would be difficult otherwise (*e.g.*, by considering other monomials).

Another consequence of implementing an exhaustive search is that we found many redundant superpolys, *i.e.*, superpolys that are identical or just linearly dependent with the ones composing the maximal rank matrix $\tilde{A}$. A similar finding is extremely interesting, because we expect it to provide a wide choice of different IV combinations yielding superpolys that compose a maximal rank sub-matrix $\tilde{A}$, thus weakening the standard assumption that cube attacks require a completely tweakable IV.

5 Related Work

The cube attack is a widely applicable method of cryptanalysis introduced by Dinur and Shamir [10]. The underlying idea, similar to Vielhaber's AIDA [24], can be extended, *e.g.*, by assigning a *dynamic* value to IV bits not belonging to the tested cube [3,11], or by replacing cubes with generic subspaces of the IV space [25], and it is used in so-called *cube testers* to detect nonrandom behaviour rather than performing key extraction [4,5]. Despite the cube attack and its variants have shown promising results against several ciphers (*e.g.*, Trivium [10], Grain [11], Hummingbird-2 [13], Katan and Simon [3], Quavium [26]), Bernstein [6] expressed harsh criticism to the feasibility and convenience of cube attacks. Indeed, a general trend for cube attacks is to focus on reduced-round variants of a cipher, without any evidence that the full version can be equally attacked. However, while Bernstein suggests that the cube attack only works if the ANF of the cipher has low degree, Fouque and Vannet [14] argue (and, to some extent, experimentally show) that effective cube attacks can be run not aiming at the maximum degree of the ANF, but rather exploiting a nonrandom ANF by searching for maxterms of significantly lower degree. Along this line, O'Neil [18] suggests that even the full version of Trivium exhibits limited randomness, thus indicating the potential vulnerability of this cipher to cube attacks.

In recent years, several implementations of the cube attack attempted at breaking Trivium, our target cipher described in Sect. A. Quedenfeld and Wolf [19] found cubes for Trivium up to round 446. Srinivasan et al. [23] introduces a sufficient condition for testing a superpoly for linearity in $\mathbb{F}_2$ with a time complexity $\mathrm{O}(2^{c+1}(k^2 + k))$, yielding 69 extremely sparse linearly independent superpolys for Trivium reduced to 576 rounds. In their seminal paper [10], Dinur

and Shamir found 63, 53, and 35 linearly independent superpolys after, respectively, 672, 735, and 767 rounds. Fouque and Vannet [14] even improve over Dinur and Shamir, by obtaining 42 linearly independent superpolys after 784 rounds, and 12 linearly independent superpolys (plus 6 quadratic superpolys) after 799 rounds. To the best of our knowledge, these are the best results against Trivium to date, making our attack comparable to (or better than) the state-of-the-art.

Distributed computing and/or parallel processing have been explored in the literature to render attacks to crypto systems computationally or storage-wise feasible/practical. Smart et al. [15] develop a new methodology to assess cryptographic key strength using cloud computing. Marks et al. [16] provide numerical evidence of the potential of mixed GPU(AMD, Nvidia) and CPU technology to data encryption and decryption algorithms. Focusing on GPU, Milo et al. [17] leverage GPUs to quickly test passphrases used to protect private keyrings of OpenPGP cryptosystems, showing that the time complexity of the attack can be reduced up to three-orders of magnitude with respect to a standard procedure, and up to ten times with respect to a highly tuned CPU implementation. A relevant result is obtained by Agostini [2] leveraging GPUs to speed up Dictionary Attacks to the BitLocker technology commonly used in Windows OSes to encrypt disks. Finally, and most closely related to the present work, Fan and Gong [13] make use of GPUs to perform side channel cube attacks on Hummingbird-2. They describe an efficient term-by-term quadraticity test for extracting simple quadratic equations, leveraging the cube attack. Just like us, Fan and Gong speed-up the implementation of the proposed term-by-term quadraticity test by leveraging GPUs and finally recovering 48 out of 128 key bits of the Hummingbird-2 with a data complexity of about 2^{18} chosen plaintexts. However, we present a complete implementation of the cube attack thoroughly designed and optimized for GPUs. Our flexible construction allows an exhaustive exploration of subsets of IV bits, thus overcoming the limitations of dynamic cube attacks, which try to find the most suitable assignment to those bits by analyzing the target cipher.

6 Conclusions and Future Work

This work has discussed in depth an advanced GPU implementation of the cube attack aimed at breaking a reduced-round version of Trivium. The implemented attack allows extending the quest for superpolys to a dimension never explored in previous works, and weakens the previous cube attack assumption of a completely tweakable IV. An extensive experimental campaign is discussed and results validate the approach and improve over the state-of-the-art attacks against reduced-round versions of Trivium.

The tool, that we expect to release into the public domain, opens new perspectives by allowing a more comprehensive and hopefully exhaustive analysis of stream-ciphers security. For instance, along the line proposed in [1], we envisage developing our implementation to test the effectiveness of the generalized cube attack over $\mathbb{F}_n$.

A Trivium Specifications

Trivium [8] is a synchronous stream cipher conceived by Christophe De Cannière and Bart Preneel, not patented, and specified as an International Standard under ISO/IEC 29192-3. Trivium combines a flexible trade-off between speed and gate count in hardware, and a reasonably efficient software implementation. Citing [9]: "Trivium is a hardware oriented design focussed on flexibility. It aims to be compact in environments with restrictions on the gate count, power-efficient on platforms with limited power resources, and fast in applications that require high-speed encryption". Particularly interesting is the fact that any state bit stays unused for at least 64 iterations after it has been modified. This means that up to 64 iterations can be parallelized and computed at once, allowing for a factor 64 reduction in the clock frequency without affecting the throughput.

Trivium generates up to 2^{64} bits of output from an 80-bit key K and an 80-bit Initial Vector IV, and it shows remarkable resistance to cryptanalysis despite its simplicity and its excellent performance. The 80-bit key K and the 80-bit IV, are used in Trivium to initialize three FSRs of length 93, 84 and 111, respectively. The internal states of the three registers are denoted $(s_1, \ldots, s_{93})$, $(s_{94}, \ldots, s_{177})$ and $(s_{178}, \ldots, s_{288})$ respectively. Fifteen out the 288 internal state bits are used at each round to compute the feedbacks to the three registers and the output bit of the cipher. However, to obtain a better mixing of the seed and to guarantee that each output bit is a complex non-linear function of all key-bits and IV-bits, the cipher undergoes 1152 initialization rounds without producing any output. In detail, the initial seed of the three registers is defined as follows:

$$\begin{aligned}(s_1, \ldots, s_{93}) &\leftarrow (K_1, \ldots, K_{80}, 0, \ldots, 0)\\ (s_{94}, \ldots, s_{177}) &\leftarrow (IV_1, \ldots, IV_{80}, 0, \ldots, 0)\\ (s_{178}, \ldots, s_{288}) &\leftarrow (0, \ldots, 0, 1, 1, 1)\end{aligned}$$

For each $t \geq 1$, the internal state of the cipher is updated as follows[4]:

$$\begin{aligned}(s_1, s_2, \ldots, s_{93}) &\leftarrow (s_{243} + s_{286} \cdot s_{287} + s_{288} + s_{69}, s_1, \ldots, s_{92})\\ (s_{94}, s_{95}, \ldots, s_{177}) &\leftarrow (s_{66} + s_{91} \cdot s_{92} + s_{93} + s_{171}, s_{94}, \ldots, s_{176})\\ (s_{178}, s_{179}, \ldots, s_{288}) &\leftarrow (s_{162} + s_{175} \cdot s_{176} + s_{177} + s_{264}, s_{178}, \ldots, s_{287})\end{aligned}$$

Finally, for each $t > 1152$, the output bit z_t is computed as:

$$z_t \leftarrow s_{66} + s_{93} + s_{162} + s_{177} + s_{243} + s_{288}$$

[4] Symbols + and · denote sum and product over $\mathbb{F}_2$, *i.e.*, bitwise XOR and AND.

B Tables of Maxterms and Superpolys

Trivium-781

maxterm bits	superpoly	round
3, 6, 8, 10, 12, 14, 18, 19, 20, 23, 25, 27, 31, 33, 38, 40, 43, 45, 48, 53, 54, 56, 58, 60, 62, 63, 69, 75, 77, 79, 80	x_{55}	781
1, 5, 7, 8, 10, 15, 16, 18, 20, 23, 25, 27, 32, 33, 36, 38, 40, 41, 43, 47, 49, 52, 53, 54, 56, 58, 63, 69, 71, 75, 77, 80	x_{69}	781
1, 6, 7, 8, 10, 12, 16, 19, 21, 24, 25, 27, 31, 33, 36, 38, 40, 41, 43, 47, 49, 52, 53, 56, 58, 63, 67, 69, 71, 73, 77, 80	x_{60}	781
1, 2, 3, 5, 6, 7, 8, 12, 14, 15, 16, 19, 21, 23, 25, 27, 36, 38, 40, 43, 45, 47, 49, 54, 56, 58, 60, 62, 69, 71, 73, 74, 80	$x_{51} + 1$	781
1, 2, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 23, 25, 27, 36, 38, 40, 43, 45, 47, 52, 54, 56, 58, 60, 62, 69, 71, 73, 76, 79, 80	x_{45}	781
1, 2, 5, 6, 7, 8, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 43, 45, 47, 49, 52, 54, 56, 58, 62, 65, 69, 71, 73, 76, 80	$x_{43} + x_{58}$	781
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 23, 27, 33, 36, 38, 40, 43, 45, 47, 52, 54, 56, 58, 60, 62, 69, 71, 73, 74, 79, 80	x_{23}	781
1, 3, 6, 7, 8, 10, 12, 14, 16, 19, 21, 23, 25, 33, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 62, 63, 65, 69, 71, 75, 77, 79, 80	$x_8 + x_{35} + x_{64}$	781
1, 5, 7, 8, 10, 12, 14, 15, 16, 18, 20, 23, 24, 25, 27, 32, 33, 36, 40, 41, 43, 47, 49, 52, 53, 56, 58, 63, 69, 71, 75, 77, 80	$x_{67} + 1$	781
3, 5, 6, 8, 10, 12, 14, 16, 18, 19, 20, 23, 24, 25, 27, 31, 33, 38, 43, 45, 48, 53, 54, 56, 58, 60, 62, 63, 69, 75, 77, 79, 80	x_2	781
6, 8, 10, 12, 14, 15, 16, 18, 19, 20, 21, 23, 25, 27, 29, 31, 33, 36, 38, 40, 41, 42, 45, 49, 54, 56, 60, 62, 63, 69, 73, 75, 80	x_{58}	781
1, 2, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 23, 27, 36, 38, 40, 43, 45, 47, 49, 52, 54, 56, 58, 60, 62, 69, 71, 73, 74, 76, 79, 80	$x_{62} + 1$	781
1, 3, 5, 6, 8, 10, 12, 14, 15, 16, 19, 21, 25, 30, 31, 32, 33, 36, 38, 40, 41, 43, 45, 47, 49, 53, 54, 56, 63, 69, 71, 73, 75, 80	$x_3+x_{25}+x_{39}+x_{40}+x_{51}+x_{66}+x_{67}+x_{78}+1$	781
1, 3, 5, 6, 8, 12, 14, 15, 16, 19, 21, 25, 27, 31, 33, 38, 40, 41, 43, 45, 47, 48, 49, 50, 53, 54, 56, 60, 63, 69, 71, 73, 75, 80	$x_{10} + x_{13} + x_{14} + x_{19} + x_{25} + x_{28} + x_{29} + x_{31} + x_{37} + x_{40} + x_{46} + x_{52} + x_{53} + x_{55} + x_{56} + x_{57} + x_{60} + x_{61} + x_{62} + x_{64} + x_{66} + x_{68} + x_{69} + 1$	781
1, 3, 5, 7, 12, 14, 15, 16, 18, 19, 20, 21, 24, 25, 27, 31, 33, 36, 40, 41, 45, 49, 54, 56, 58, 60, 62, 63, 66, 71, 73, 75, 77, 80	x_{57}	781
1, 3, 6, 7, 8, 10, 12, 14, 16, 19, 21, 23, 25, 33, 36, 40, 41, 43, 45, 47, 49, 53, 54, 56, 58, 61, 63, 65, 69, 71, 75, 77, 79, 80	$x_{43} + x_{58} + x_{64} + x_{66} + x_{70}$	781
1, 3, 5, 6, 7, 8, 12, 13, 14, 15, 16, 19, 21, 23, 25, 27, 36, 38, 40, 43, 45, 47, 49, 52, 54, 56, 58, 62, 65, 69, 71, 73, 76, 79, 80	x_{65}	781
1, 3, 5, 6, 8, 10, 12, 14, 15, 16, 18, 19, 24, 25, 31, 32, 33, 36, 38, 40, 41, 43, 45, 47, 49, 50, 53, 54, 56, 63, 69, 71, 73, 75, 80	$x_{23} + x_{39} + x_{50} + x_{66} + x_{67} + x_{79} + 1$	781
1, 3, 5, 6, 8, 10, 12, 14, 15, 16, 18, 19, 25, 30, 31, 32, 33, 38, 40, 41, 43, 45, 47, 49, 50, 53, 54, 56, 60, 63, 69, 71, 73, 75, 80	$x_9+x_{18}+x_{24}+x_{26}+x_{32}+x_{33}+x_{34}+x_{42}+x_{51}+x_{53}+x_{54}+x_{58}+x_{59}+x_{64}+x_{66}+x_{68}+x_{69}+x_{80}+1$	781
1, 3, 5, 6, 8, 10, 12, 14, 15, 16, 19, 21, 24, 25, 31, 32, 33, 36, 38, 40, 41, 43, 45, 47, 49, 50, 53, 54, 56, 63, 69, 71, 73, 75, 80	$x_{52} + x_{66} + x_{67} + x_{79}$	781
1, 3, 5, 6, 8, 12, 14, 15, 16, 18, 19, 21, 25, 27, 31, 33, 38, 40, 41, 43, 45, 47, 48, 49, 50, 53, 54, 56, 60, 63, 69, 71, 73, 75, 80	$x_{13} + x_{14} + x_{19} + x_{25} + x_{27} + x_{28} + x_{29} + x_{31} + x_{39} + x_{41} + x_{42} + x_{46} + x_{51} + x_{52} + x_{54} + x_{55} + x_{56} + x_{57} + x_{61} + x_{62} + x_{64} + x_{65} + x_{66} + x_{69} + x_{78}$	781
1, 3, 5, 6, 8, 12, 14, 15, 16, 18, 19, 24, 25, 27, 31, 32, 33, 36, 38, 40, 41, 45, 47, 48, 49, 53, 54, 56, 60, 63, 69, 71, 73, 75, 80	$x_{16} + x_{26} + x_{27} + x_{38} + x_{43} + x_{53} + x_{54} + x_{56} + x_{65} + x_{67} + x_{80}$	781
1, 3, 5, 6, 8, 12, 14, 15, 16, 19, 21, 24, 25, 30, 31, 32, 33, 36, 38, 40, 41, 45, 47, 52, 54, 56, 58, 60, 62, 65, 69, 70, 71, 73, 74, 76, 80	$x_{25} + x_{27} + x_{30} + x_{54} + x_{57}$	781
1, 2, 3, 5, 6, 7, 8, 12, 14, 15, 16, 19, 23, 25, 27, 33, 36, 38, 40, 43, 45, 47, 52, 54, 56, 58, 60, 62, 65, 69, 70, 71, 73, 74, 76, 80	x_{42}	781
1, 3, 5, 6, 8, 10, 12, 14, 15, 16, 18, 19, 21, 25, 27, 30, 31, 32, 33, 38, 40, 41, 43, 45, 47, 48, 49, 53, 54, 56, 63, 69, 71, 73, 75, 80	$x_{14} + x_{29} + x_{41} + x_{55} + x_{61} + x_{62}$	781
1, 3, 5, 6, 8, 10, 12, 14, 15, 16, 18, 19, 24, 25, 31, 32, 33, 36, 38, 40, 41, 43, 45, 47, 48, 49, 50, 53, 54, 56, 63, 69, 71, 73, 75, 80	$x_{39} + x_{66}$	781
1, 3, 5, 6, 8, 10, 12, 14, 15, 16, 18, 19, 25, 30, 31, 32, 33, 38, 40, 41, 43, 45, 47, 48, 49, 50, 53, 54, 56, 60, 63, 69, 71, 73, 75, 80	$x_{24} + x_{55} + x_{61} + x_{66} + x_{67} + 1$	781
1, 3, 5, 6, 8, 10, 12, 14, 15, 16, 19, 21, 24, 25, 30, 31, 33, 38, 40, 41, 43, 45, 47, 49, 50, 53, 54, 56, 60, 63, 69, 70, 71, 73, 75, 80	$x_{12}+x_{27}+x_{32}+x_{33}+x_{40}+x_{42}+x_{51}+x_{53}+x_{57}+x_{58}+x_{60}+x_{64}+x_{80}+1$	781
1, 3, 5, 6, 8, 10, 12, 14, 15, 16, 19, 21, 24, 25, 30, 31, 33, 38, 40, 41, 43, 45, 47, 49, 50, 53, 54, 56, 60, 63, 69, 70, 71, 73, 75, 80	$x_{27} + x_{32} + x_{42} + x_{53} + x_{58} + x_{60} + x_{64} + x_{78} + x_{80} + 1$	781
1, 3, 5, 6, 8, 12, 14, 15, 16, 18, 19, 24, 25, 27, 30, 31, 32, 33, 38, 40, 41, 45, 47, 49, 50, 53, 54, 56, 60, 63, 69, 70, 71, 73, 75, 80	$x_{11} + x_{24} + x_{25} + x_{29} + x_{30} + x_{31} + x_{40} + x_{41} + x_{45} + x_{50} + x_{52} + x_{53} + x_{54} + x_{56} + x_{58} + x_{61} + x_{65} + x_{66} + x_{67} + x_{68} + x_{77} + x_{79} + x_{80} + 1$	781

1, 3, 5, 6, 8, 12, 14, 15, 16, 19, 21, 24, 25, 30, 31, 32, 33, 36, 38, 40, 41, 43, 45, 47, 49, 50, 53, 54, 56, 60, 63, 69, 71, 73, 75, 80	$x_{14} + x_{16} + x_{27} + x_{29} + x_{30} + x_{31} + x_{40} + x_{41} + x_{42} + x_{43} + x_{54} + x_{55} + x_{56} + x_{57} + x_{58} + x_{64} + x_{79} + x_{80} + 1$	781
1, 3, 5, 6, 8, 12, 14, 15, 16, 19, 21, 24, 25, 30, 31, 32, 33, 36, 38, 40, 41, 43, 45, 47, 49, 53, 54, 56, 60, 63, 69, 70, 71, 73, 75, 80	$x_{28} + x_{29} + x_{32} + x_{33} + x_{40} + x_{41} + x_{42} + x_{44} + x_{50} + x_{51} + x_{55} + x_{56} + x_{57} + x_{59} + x_{61} + x_{62} + x_{64} + x_{66} + x_{67} + x_{68} + x_{70} + x_{78} + 1$	781
1, 3, 5, 6, 8, 12, 14, 15, 16, 19, 21, 24, 25, 30, 31, 33, 36, 38, 40, 41, 43, 45, 47, 49, 50, 53, 54, 56, 60, 63, 69, 70, 71, 73, 75, 80	$x_{38} + x_{39} + x_{41} + x_{44} + x_{45} + x_{50} + x_{51} + x_{52} + x_{53} + x_{55} + x_{57} + x_{58} + x_{60} + x_{66} + x_{68} + x_{72} + x_{78} + x_{79} + 1$	781
1, 3, 5, 6, 8, 12, 14, 15, 16, 19, 21, 24, 25, 31, 33, 36, 38, 40, 41, 43, 45, 47, 48, 49, 50, 53, 54, 56, 60, 63, 69, 70, 71, 73, 75, 80	$x_{43} + x_{50} + x_{52} + x_{55} + x_{58} + x_{66} + x_{70} + x_{77} + 1$	781
1, 5, 6, 8, 12, 14, 15, 16, 18, 19, 24, 25, 27, 30, 31, 32, 33, 38, 40, 41, 43, 45, 47, 49, 50, 53, 54, 56, 60, 63, 69, 70, 71, 73, 75, 80	$x_{41} + x_{53} + x_{55} + x_{58} + x_{61} + x_{68}$	781
1, 5, 6, 8, 12, 14, 15, 16, 18, 19, 24, 25, 27, 30, 31, 32, 33, 38, 40, 41, 43, 45, 47, 49, 50, 53, 54, 56, 60, 63, 69, 70, 71, 73, 75, 80	$x_{29} + x_{41} + x_{42} + x_{53} + x_{55} + x_{56} + x_{58} + x_{61} + x_{64} + x_{66} + x_{67} + x_{68} + x_{69}$	781
1, 5, 6, 8, 12, 14, 15, 16, 18, 19, 24, 25, 27, 30, 31, 32, 33, 38, 40, 41, 43, 45, 47, 49, 50, 53, 54, 56, 60, 63, 69, 70, 71, 73, 75, 80	$x_{14} + x_{55} + x_{58} + x_{61} + x_{64} + x_{66} + x_{68} + x_{80}$	781
1, 5, 6, 10, 12, 14, 15, 18, 19, 20, 21, 22, 23, 25, 27, 28, 29, 31, 33, 36, 38, 40, 41, 42, 45, 48, 49, 54, 60, 62, 63, 69, 73, 75, 77, 80	x_{64}	781
5, 6, 8, 10, 12, 14, 15, 16, 18, 19, 20, 21, 22, 23, 25, 27, 29, 31, 33, 36, 38, 40, 41, 42, 45, 48, 49, 54, 60, 62, 63, 69, 73, 75, 77, 80	$x_{66} + 1$	781
1, 2, 3, 5, 6, 7, 8, 12, 13, 14, 15, 16, 19, 21, 23, 27, 33, 36, 38, 40, 43, 45, 47, 52, 54, 56, 58, 60, 62, 69, 70, 71, 73, 74, 76, 79, 80	x_{56}	781
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 33, 36, 40, 43, 45, 47, 49, 53, 54, 56, 58, 62, 63, 65, 69, 71, 74, 75, 77, 78, 79, 80	$x_{21} + x_{36} + x_{48} + x_{58} + x_{63} + 1$	781
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 33, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 62, 63, 65, 69, 71, 74, 75, 77, 78, 79, 80	$x_{19}+x_{27}+x_{45}+x_{54}+x_{64}+x_{66}+x_{72}+1$	781
1, 3, 5, 6, 8, 12, 14, 15, 16, 18, 19, 24, 25, 30, 31, 32, 33, 36, 38, 40, 41, 43, 45, 47, 49, 50, 53, 54, 56, 60, 63, 69, 70, 71, 73, 75, 80	$x_5 + x_8 + x_{24} + x_{26} + x_{32} + x_{33} + x_{39} + x_{40} + x_{41} + x_{42} + x_{44} + x_{47} + x_{51} + x_{54} + x_{57} + x_{59} + x_{60} + x_{65} + x_{66} + x_{68} + x_{69} + x_{78} + x_{79}$	781
1, 3, 5, 6, 8, 12, 14, 15, 16, 19, 21, 25, 27, 31, 32, 33, 36, 38, 40, 41, 45, 47, 49, 50, 53, 54, 56, 60, 63, 69, 71, 73, 75, 80	$x_{25} + x_{52} + 1$	782
1, 5, 6, 7, 8, 10, 12, 14, 15, 19, 21, 24, 25, 27, 31, 36, 38, 39, 40, 41, 45, 47, 49, 53, 56, 58, 62, 63, 66, 69, 71, 73, 77, 80	x_{40}	782
1, 3, 5, 6, 7, 10, 12, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 43, 45, 47, 49, 53, 56, 58, 61, 62, 63, 65, 69, 71, 74, 75, 77, 78	$x_{25} + 1$	782
1, 3, 5, 6, 10, 12, 14, 15, 16, 18, 19, 21, 25, 27, 31, 32, 33, 38, 40, 41, 43, 45, 47, 48, 49, 50, 53, 54, 56, 60, 63, 69, 71, 73, 75, 80	$x_{13} + x_{16} + x_{19} + x_{25} + x_{29} + x_{33} + x_{35} + x_{36} + x_{37} + x_{38} + x_{39} + x_{40} + x_{42} + x_{45} + x_{51} + x_{52} + x_{53} + x_{54} + x_{55} + x_{62} + x_{63} + x_{64} + x_{65} + x_{67} + x_{69} + x_{70} + x_{71} + x_{73} + x_{79} + x_{80} + 1$	782
5, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 34, 36, 38, 40, 45, 47, 48, 49, 53, 54, 56, 58, 60, 62, 63, 69, 71, 80	$x_{38} + 1$	783
3, 5, 6, 7, 8, 10, 14, 15, 16, 21, 23, 25, 27, 33, 34, 36, 38, 40, 45, 47, 48, 49, 53, 54, 55, 56, 58, 61, 62, 63, 69, 71, 74, 80	$x_{27} + 1$	783
1, 5, 6, 8, 10, 12, 14, 15, 16, 18, 19, 21, 24, 25, 27, 30, 31, 32, 33, 38, 40, 41, 45, 47, 48, 49, 50, 53, 54, 56, 63, 69, 71, 73, 75, 80	$x_{32} + x_{49} + x_{52} + x_{56} + x_{59} + x_{61} + x_{62} + x_{79} + 1$	783
1, 5, 6, 8, 10, 12, 14, 15, 16, 18, 19, 21, 24, 25, 31, 33, 36, 38, 40, 41, 43, 45, 47, 48, 49, 50, 53, 54, 56, 60, 63, 69, 71, 73, 75, 80	$x_7+x_{16}+x_{40}+x_{43}+x_{49}+x_{52}+x_{58}+x_{62}+x_{70}+x_{79}+1$	783
1, 3, 5, 6, 8, 10, 12, 14, 15, 16, 18, 19, 24, 25, 30, 31, 32, 33, 36, 38, 40, 41, 43, 45, 47, 49, 50, 53, 54, 56, 60, 63, 69, 71, 73, 75, 80	$x_{26} + x_{66} + x_{68} + 1$	783
1, 3, 6, 7, 10, 12, 15, 16, 19, 21, 23, 25, 33, 36, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 61, 63, 65, 69, 71, 75, 77, 80	x_4	784
1, 5, 6, 7, 8, 10, 12, 14, 15, 16, 21, 23, 25, 27, 33, 36, 38, 40, 43, 45, 47, 49, 53, 54, 55, 56, 58, 60, 62, 63, 67, 69, 71, 80	$x_{53} + 1$	784
1, 6, 8, 10, 12, 14, 15, 16, 18, 19, 20, 21, 22, 23, 25, 27, 29, 33, 36, 38, 40, 41, 42, 45, 48, 49, 54, 60, 62, 63, 69, 73, 75, 80	x_{37}	784
3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 60, 62, 63, 69, 71, 74, 80	x_{36}	784
1, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 29, 31, 33, 36, 38, 40, 41, 45, 47, 49, 53, 58, 60, 63, 71, 75, 76, 80	$x_{12} + 1$	785
1, 3, 6, 7, 10, 12, 14, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 63, 65, 69, 71, 74, 75, 77, 78, 80	x_{34}	785
1, 5, 6, 7, 8, 12, 13, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 43, 45, 47, 49, 52, 54, 56, 58, 60, 62, 69, 71, 73, 79, 80	x_{54}	785
1, 3, 5, 6, 7, 10, 12, 14, 16, 19, 21, 23, 25, 27, 33, 36, 40, 43, 45, 47, 49, 53, 54, 56, 58, 61, 62, 63, 69, 71, 74, 75, 77, 78, 80	$x_{13} + x_{55} + x_{60} + x_{64}$	785
1, 3, 5, 6, 7, 10, 12, 14, 16, 19, 21, 23, 25, 27, 33, 36, 40, 41, 43, 45, 47, 49, 53, 54, 56, 58, 62, 63, 65, 69, 71, 74, 75, 77, 78, 80	$x_{22} + x_{49} + x_{64}$	786
1, 3, 6, 7, 10, 12, 14, 16, 19, 21, 23, 25, 27, 33, 36, 40, 41, 43, 45, 47, 49, 53, 54, 56, 58, 61, 62, 63, 65, 69, 71, 74, 75, 77, 78, 80	$x_{14} + x_{23} + x_{41} + x_{47} + x_{49} + x_{50} + x_{58} + x_{64}$	786
1, 3, 5, 6, 7, 8, 10, 12, 14, 16, 19, 21, 23, 25, 27, 33, 36, 40, 43, 45, 47, 49, 53, 54, 56, 58, 61, 62, 63, 65, 69, 71, 74, 75, 77, 78, 80	$x_3 + x_4 + x_{20} + x_{22} + x_{30} + x_{34} + x_{38} + x_{40} + x_{42} + x_{45} + x_{49} + x_{51} + x_{58} + x_{61} + x_{65} + x_{67} + x_{69} + x_{72} + x_{78}$	786
1, 3, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 38, 40, 43, 45, 47, 49, 53, 56, 58, 62, 63, 69, 71, 75, 77, 79	$x_9+x_{29}+x_{30}+x_{32}+x_{42}+x_{43}+x_{49}+x_{51}+x_{57}+x_{58}+x_{59}+x_{60}+x_{62}+x_{64}+x_{66}+x_{67}+x_{68}+x_{69}+x_{70}+x_{72}+x_{76}$	791
1, 3, 6, 7, 8, 10, 12, 15, 16, 19, 21, 23, 25, 27, 33, 36, 40, 41, 43, 45, 47, 49, 53, 56, 58, 62, 63, 69, 71, 75, 77, 80	$x_{17}+x_{26}+x_{30}+x_{32}+x_{41}+x_{43}+x_{47}+x_{57}+x_{62}+x_{65}+x_{66}+x_{70}+x_{72}+x_{74}+1$	791

1, 3, 6, 7, 8, 10, 12, 15, 16, 19, 21, 23, 25, 27, 33, 36, 40, 41, 43, 45, 47, 49, 53, 56, 58, 62, 63, 69, 71, 75, 77, 80	$x_{14} + x_{17} + x_{26} + x_{30} + x_{43} + x_{47} + x_{50} + x_{57} + x_{58} + x_{59} + x_{65} + x_{70} + x_{72} + x_{74} + x_{77} + 1$	791
1, 3, 5, 6, 7, 8, 10, 12, 14, 16, 19, 21, 23, 25, 27, 33, 36, 40, 41, 43, 45, 47, 49, 53, 56, 58, 63, 65, 69, 71, 75, 77, 79	$x_{12}+x_{26}+x_{30}+x_{39}+x_{41}+x_{45}+x_{47}+x_{57}+x_{58}+x_{59}+x_{62}+x_{64}+x_{74}+x_{76}+1$	791
1, 3, 5, 6, 7, 8, 10, 12, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 41, 43, 45, 47, 49, 53, 56, 58, 62, 63, 65, 69, 71, 75, 77	$x_5 + x_{18} + x_{20} + x_{26} + x_{28} + x_{29} + x_{30} + x_{31} + x_{32} + x_{41} + x_{42} + x_{44} + x_{50} + x_{51} + x_{56} + x_{57} + x_{62} + x_{64} + x_{67} + x_{69} + x_{70} + x_{71} + x_{74} + x_{77} + x_{78} + 1$	791
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 38, 40, 41, 43, 45, 47, 49, 53, 56, 58, 61, 62, 63, 65, 69, 71, 75, 77	$x_1+x_{28}+x_{32}+x_{47}+x_{58}+x_{59}+x_{62}+x_{64}+x_{74}$	791
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 41, 43, 45, 47, 49, 53, 56, 58, 61, 62, 63, 65, 69, 71, 75, 77, 80	$x_{10} + x_{11} + x_{12} + x_{13} + x_{15} + x_{17} + x_{19} + x_{20} + x_{29} + x_{31} + x_{32} + x_{33} + x_{37} + x_{39} + x_{40} + x_{41} + x_{42} + x_{44} + x_{46} + x_{48} + x_{49} + x_{50} + x_{53} + x_{57} + x_{60} + x_{67} + x_{70} + x_{71} + x_{76} + x_{78} + x_{79}$	791
1, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 29, 31, 33, 36, 38, 40, 41, 42, 45, 47, 49, 53, 58, 63, 69, 71, 72, 76, 79, 80	x_{61}	791
1, 3, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 33, 36, 38, 40, 41, 43, 45, 47, 49, 53, 54, 56, 58, 62, 63, 69, 71, 74, 75, 77, 78, 80	$x_{43} + x_{47} + x_{58} + x_{70} + x_{74} + 1$	791
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 38, 40, 41, 43, 45, 47, 49, 53, 56, 58, 62, 63, 69, 71, 75, 77, 80	$x_{12} + x_{17} + x_{26} + x_{27} + x_{29} + x_{30} + x_{32} + x_{40} + x_{43} + x_{45} + x_{46} + x_{49} + x_{53} + x_{54} + x_{56} + x_{59} + x_{62} + x_{64} + x_{65} + x_{67} + x_{69} + x_{72} + x_{74} + x_{75}$	792
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 41, 43, 45, 47, 49, 53, 56, 58, 61, 63, 69, 71, 75, 77, 78, 79, 80	$x_{12}+x_{14}+x_{26}+x_{30}+x_{40}+x_{41}+x_{47}+x_{48}+x_{56}+x_{66}+x_{67}+x_{68}+x_{74}+x_{75}+1$	792
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 41, 43, 45, 47, 49, 53, 54, 56, 58, 63, 69, 71, 74, 75, 77, 78, 79, 80	$x_{16} + x_{43} + x_{56}$	792
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 61, 63, 65, 69, 71, 75, 77, 78, 79, 80	$x_{14} + x_{16} + x_{26} + x_{29} + x_{30} + x_{41} + x_{45} + x_{55} + x_{56} + x_{59} + x_{62} + x_{64} + x_{66} + x_{68} + x_{70} + x_{71} + x_{72} + 1$	792
1, 3, 6, 7, 8, 10, 12, 14, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 41, 43, 45, 47, 49, 53, 54, 56, 58, 61, 63, 69, 71, 75, 77, 80	$x_{45} + x_{72}$	793
1, 6, 7, 8, 10, 12, 14, 16, 19, 21, 23, 25, 27, 30, 31, 33, 38, 40, 43, 45, 47, 49, 51, 52, 56, 58, 63, 67, 69, 71, 73, 77, 80	$x_{10} + x_{55}$	798
1, 3, 5, 6, 7, 8, 10, 12, 15, 16, 19, 21, 23, 25, 27, 33, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 62, 63, 65, 69, 71, 74, 75, 77, 80	$x_{36} + x_{52} + x_{60} + x_{63}$	798
1, 3, 5, 6, 7, 8, 10, 12, 15, 16, 19, 21, 23, 25, 27, 33, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 61, 62, 63, 65, 69, 71, 74, 75, 77, 80	$x_6+x_{11}+x_{25}+x_{33}+x_{36}+x_{53}+x_{60}+x_{62}+x_{63}+x_{64}+x_{79}$	798

Trivium-784

maxterm bits	superpoly	round
1, 3, 6, 7, 10, 12, 15, 16, 19, 21, 23, 25, 33, 36, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 61, 63, 65, 69, 71, 75, 77, 80	x_4	784
1, 5, 6, 8, 10, 12, 14, 15, 16, 18, 19, 20, 21, 23, 25, 27, 29, 31, 33, 36, 38, 40, 41, 42, 45, 49, 54, 60, 62, 69, 73, 75, 80	x_{60}	784
5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 61, 62, 63, 67, 69, 71, 80	$x_{56} + 1$	784
1, 3, 5, 6, 7, 10, 12, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 61, 63, 65, 69, 71, 75, 77, 78	$x_2 + x_9 + x_{13} + x_{14} + x_{22} + x_{23} + x_{30} + x_{36} + x_{38} + x_{39} + x_{40} + x_{42} + x_{47} + x_{48} + x_{51} + x_{56} + x_{65} + x_{67} + x_{68} + x_{69} + x_{74} + x_{75}$	784
1, 3, 5, 6, 8, 10, 12, 14, 15, 16, 19, 21, 25, 30, 31, 32, 33, 38, 40, 41, 43, 45, 47, 48, 49, 50, 53, 54, 56, 63, 69, 71, 75, 80	x_{38}	784
1, 3, 6, 7, 10, 12, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 41, 43, 45, 47, 49, 53, 54, 56, 58, 61, 63, 69, 71, 75, 77, 78, 79	$x_{38} + x_{47} + x_{74}$	784
1, 5, 6, 7, 8, 10, 12, 14, 15, 16, 21, 23, 25, 27, 33, 36, 38, 40, 43, 45, 47, 49, 53, 54, 55, 56, 58, 60, 62, 63, 67, 69, 71, 80	$x_{53} + 1$	784
1, 5, 6, 7, 8, 10, 12, 14, 15, 16, 21, 23, 25, 27, 33, 36, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 61, 62, 63, 67, 69, 71, 74, 80	x_{58}	784
1, 6, 8, 10, 12, 14, 15, 16, 18, 19, 20, 21, 22, 23, 25, 27, 29, 33, 36, 38, 40, 41, 42, 45, 48, 49, 54, 60, 62, 63, 69, 73, 75, 80	x_{37}	784
1, 6, 8, 10, 12, 14, 15, 16, 18, 19, 20, 21, 22, 23, 25, 27, 29, 33, 36, 38, 40, 41, 45, 48, 49, 54, 56, 60, 62, 69, 73, 75, 77, 80	x_{64}	784
3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 60, 62, 63, 69, 71, 74, 80	x_{36}	784
6, 8, 10, 12, 14, 15, 16, 18, 19, 20, 21, 22, 23, 25, 27, 29, 33, 36, 38, 40, 41, 45, 48, 49, 54, 56, 60, 62, 63, 69, 73, 75, 77, 80	x_{66}	784
5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 34, 36, 38, 40, 43, 45, 47, 49, 53, 54, 55, 56, 58, 60, 62, 63, 69, 71, 74, 80	$x_{67} + 1$	784
5, 6, 8, 10, 12, 14, 15, 18, 19, 20, 21, 22, 23, 25, 27, 28, 29, 31, 33, 36, 38, 40, 41, 42, 45, 49, 54, 60, 62, 63, 69, 73, 75, 77, 80	x_{62}	784
1, 5, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 34, 36, 38, 40, 43, 45, 47, 48, 49, 53, 54, 56, 58, 60, 61, 62, 63, 69, 71, 74, 80	$x_{69} + 1$	784
3, 5, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 43, 45, 47, 48, 49, 53, 54, 56, 58, 60, 61, 62, 63, 67, 69, 71, 74, 80	x_{40}	784
1, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 29, 31, 33, 36, 38, 40, 41, 45, 47, 49, 53, 58, 60, 63, 71, 75, 76, 80	$x_{12} + 1$	785
1, 5, 6, 7, 10, 12, 16, 19, 21, 23, 24, 25, 27, 31, 33, 36, 38, 40, 41, 43, 47, 49, 52, 56, 58, 60, 63, 67, 69, 71, 73, 77, 80	x_{42}	785

1, 5, 6, 10, 12, 14, 15, 16, 19, 21, 25, 27, 30, 31, 33, 36, 38, 40, 41, 43, 45, 47, 48, 49, 53, 54, 56, 63, 69, 71, 73, 75, 80	x_{55}	785
1, 3, 6, 7, 10, 12, 14, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 63, 65, 69, 71, 74, 75, 77, 78, 80	x_{34}	785
1, 5, 6, 7, 8, 12, 13, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 43, 45, 47, 49, 52, 54, 56, 58, 60, 62, 69, 71, 73, 79, 80	x_{54}	785
1, 3, 5, 6, 7, 10, 12, 14, 16, 19, 21, 23, 25, 27, 33, 36, 40, 43, 45, 47, 49, 53, 54, 56, 58, 61, 62, 63, 69, 71, 74, 75, 77, 78, 80	$x_{13} + x_{55} + x_{60} + x_{64}$	785
1, 3, 5, 6, 7, 10, 12, 14, 16, 19, 21, 23, 25, 27, 33, 36, 40, 41, 43, 45, 47, 49, 53, 54, 56, 58, 62, 63, 65, 69, 71, 74, 75, 77, 78, 80	$x_{22} + x_{49} + x_{64}$	786
1, 3, 6, 7, 8, 10, 12, 14, 16, 19, 21, 23, 25, 27, 33, 36, 40, 43, 45, 47, 49, 53, 54, 56, 58, 61, 62, 63, 65, 69, 71, 74, 75, 77, 78, 80	$x_3+x_4+x_7+x_{12}+x_{20}+x_{22}+x_{30}+x_{36}+x_{39}+x_{42}+x_{43}+x_{45}+x_{47}+x_{51}+x_{58}+x_{63}+x_{69}+x_{70}+x_{72}+x_{78}+1$	786
1, 3, 6, 7, 10, 12, 14, 16, 19, 21, 23, 25, 27, 33, 36, 40, 41, 43, 45, 47, 49, 53, 54, 56, 58, 61, 62, 63, 65, 69, 71, 74, 75, 77, 78, 80	$x_{14} + x_{23} + x_{41} + x_{47} + x_{49} + x_{50} + x_{58} + x_{64}$	786
1, 3, 5, 6, 7, 8, 10, 12, 14, 16, 19, 21, 23, 25, 27, 33, 36, 40, 43, 45, 47, 49, 53, 54, 56, 58, 61, 62, 63, 65, 69, 71, 74, 75, 77, 78, 80	$x_3+x_4+x_{14}+x_{22}+x_{30}+x_{34}+x_{38}+x_{41}+x_{42}+x_{45}+x_{47}+x_{49}+x_{51}+x_{58}+x_{61}+x_{65}+x_{69}+x_{72}+x_{78}$	786
1, 3, 5, 6, 7, 8, 10, 12, 14, 16, 19, 21, 23, 25, 27, 33, 36, 40, 43, 45, 47, 49, 53, 54, 56, 58, 62, 63, 69, 71, 75, 77, 79	$x_3+x_4+x_{20}+x_{22}+x_{30}+x_{34}+x_{38}+x_{40}+x_{42}+x_{45}+x_{49}+x_{51}+x_{58}+x_{61}+x_{65}+x_{67}+x_{69}+x_{72}+x_{78}$	786
1, 3, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 38, 40, 43, 45, 47, 49, 53, 56, 58, 62, 63, 69, 71, 75, 77, 79	$x_9+x_{29}+x_{30}+x_{32}+x_{42}+x_{43}+x_{49}+x_{51}+x_{57}+x_{58}+x_{59}+x_{60}+x_{62}+x_{64}+x_{66}+x_{67}+x_{68}+x_{69}+x_{70}+x_{72}+x_{76}$	791
1, 3, 6, 7, 8, 10, 12, 15, 16, 19, 21, 23, 25, 27, 33, 36, 40, 41, 43, 45, 47, 49, 53, 56, 58, 62, 63, 69, 71, 75, 77, 80	$x_{17}+x_{26}+x_{30}+x_{32}+x_{41}+x_{43}+x_{47}+x_{57}+x_{62}+x_{65}+x_{66}+x_{70}+x_{72}+x_{74}+1$	791
1, 3, 6, 7, 8, 10, 12, 15, 16, 19, 21, 23, 25, 27, 33, 36, 40, 41, 43, 45, 47, 49, 53, 56, 58, 62, 63, 69, 71, 75, 77, 80	$x_1+x_{17}+x_{26}+x_{28}+x_{41}+x_{43}+x_{47}+x_{49}+x_{59}+x_{62}+x_{64}+x_{65}+x_{66}+x_{70}+x_{72}+x_{74}+x_{76}+1$	791
1, 3, 6, 7, 8, 10, 12, 15, 16, 19, 21, 23, 25, 27, 33, 36, 40, 41, 43, 45, 47, 49, 53, 56, 58, 62, 63, 69, 71, 75, 77, 80	$x_{14}+x_{17}+x_{26}+x_{30}+x_{43}+x_{47}+x_{50}+x_{57}+x_{58}+x_{59}+x_{65}+x_{70}+x_{72}+x_{74}+x_{77}+1$	791
1, 3, 5, 6, 7, 8, 10, 12, 14, 16, 19, 21, 23, 25, 27, 33, 36, 40, 41, 43, 45, 47, 49, 53, 56, 58, 63, 65, 69, 71, 75, 77, 79	$x_{12}+x_{26}+x_{30}+x_{39}+x_{41}+x_{45}+x_{47}+x_{57}+x_{58}+x_{59}+x_{62}+x_{64}+x_{74}+x_{76}+1$	791
1, 3, 5, 6, 7, 8, 10, 12, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 41, 43, 45, 47, 49, 53, 56, 58, 62, 63, 65, 69, 71, 75, 77	$x_5+x_{18}+x_{20}+x_{26}+x_{28}+x_{29}+x_{30}+x_{31}+x_{32}+x_{41}+x_{42}+x_{44}+x_{50}+x_{51}+x_{56}+x_{57}+x_{62}+x_{64}+x_{67}+x_{69}+x_{70}+x_{71}+x_{74}+x_{77}+x_{78}+1$	791
1, 3, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 40, 41, 43, 45, 47, 49, 53, 56, 58, 61, 62, 63, 65, 69, 71, 75, 77, 80	$x_7+x_9+x_{10}+x_{11}+x_{12}+x_{13}+x_{15}+x_{17}+x_{19}+x_{20}+x_{26}+x_{30}+x_{31}+x_{33}+x_{37}+x_{39}+x_{40}+x_{41}+x_{42}+x_{44}+x_{45}+x_{46}+x_{47}+x_{48}+x_{50}+x_{51}+x_{53}+x_{56}+x_{59}+x_{60}+x_{64}+x_{67}+x_{68}+x_{70}+x_{71}+x_{72}+x_{74}+x_{78}+x_{79}+1$	791
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 38, 40, 41, 43, 45, 47, 49, 53, 56, 58, 61, 62, 63, 65, 69, 71, 75, 77	$x_1+x_{28}+x_{32}+x_{47}+x_{58}+x_{59}+x_{62}+x_{64}+x_{74}$	791
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 40, 41, 43, 45, 47, 49, 53, 56, 58, 61, 62, 63, 65, 69, 71, 75, 77, 80	$x_3+x_4+x_6+x_7+x_9+x_{10}+x_{13}+x_{15}+x_{19}+x_{22}+x_{28}+x_{30}+x_{33}+x_{34}+x_{35}+x_{38}+x_{39}+x_{40}+x_{41}+x_{43}+x_{44}+x_{47}+x_{48}+x_{49}+x_{50}+x_{53}+x_{54}+x_{55}+x_{56}+x_{58}+x_{61}+x_{62}+x_{65}+x_{66}+x_{67}+x_{68}+x_{69}+x_{71}+x_{72}+x_{76}+x_{77}+x_{78}$	791
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 40, 41, 43, 45, 47, 49, 53, 56, 58, 61, 62, 63, 65, 69, 71, 75, 77, 80	$x_{10}+x_{11}+x_{12}+x_{13}+x_{15}+x_{17}+x_{19}+x_{20}+x_{29}+x_{31}+x_{32}+x_{33}+x_{37}+x_{39}+x_{40}+x_{41}+x_{42}+x_{44}+x_{46}+x_{48}+x_{49}+x_{50}+x_{53}+x_{57}+x_{60}+x_{67}+x_{70}+x_{71}+x_{76}+x_{78}+x_{79}$	791
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 40, 41, 43, 45, 47, 49, 53, 56, 58, 61, 62, 63, 65, 69, 71, 75, 77, 80	$x_{10}+x_{11}+x_{12}+x_{13}+x_{14}+x_{15}+x_{17}+x_{19}+x_{20}+x_{26}+x_{29}+x_{31}+x_{33}+x_{37}+x_{39}+x_{40}+x_{42}+x_{44}+x_{46}+x_{48}+x_{53}+x_{57}+x_{58}+x_{59}+x_{60}+x_{66}+x_{67}+x_{68}+x_{70}+x_{71}+x_{72}+x_{77}+x_{78}+x_{79}+1$	791
1, 3, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 40, 41, 43, 45, 47, 49, 53, 54, 56, 58, 61, 62, 63, 65, 69, 71, 75, 77, 79	$x_3+x_4+x_6+x_{11}+x_{15}+x_{17}+x_{19}+x_{20}+x_{22}+x_{30}+x_{34}+x_{35}+x_{37}+x_{38}+x_{39}+x_{42}+x_{43}+x_{47}+x_{51}+x_{54}+x_{57}+x_{58}+x_{60}+x_{61}+x_{64}+x_{65}+x_{67}+x_{68}+x_{70}+x_{72}+x_{74}+x_{77}+x_{79}+1$	791
1, 3, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 40, 41, 43, 45, 47, 49, 53, 54, 56, 58, 61, 62, 63, 65, 69, 71, 75, 77, 79	$x_3+x_4+x_6+x_{11}+x_{12}+x_{15}+x_{17}+x_{18}+x_{19}+x_{20}+x_{22}+x_{30}+x_{34}+x_{35}+x_{37}+x_{38}+x_{39}+x_{42}+x_{43}+x_{45}+x_{47}+x_{50}+x_{54}+x_{56}+x_{57}+x_{58}+x_{61}+x_{65}+x_{69}+x_{70}+x_{74}+x_{78}+x_{79}+1$	791
1, 3, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 33, 36, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 62, 63, 69, 71, 74, 75, 77, 78, 80	$x_1+x_5+x_9+x_{14}+x_{18}+x_{20}+x_{26}+x_{28}+x_{32}+x_{41}+x_{42}+x_{43}+x_{45}+x_{47}+x_{49}+x_{66}+x_{67}+x_{69}+x_{70}+x_{76}+x_{78}$	791
1, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 29, 31, 33, 36, 38, 40, 41, 42, 45, 47, 49, 53, 58, 63, 69, 71, 72, 76, 79, 80	x_{61}	791
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 40, 41, 43, 45, 47, 49, 53, 54, 56, 58, 61, 62, 63, 65, 69, 71, 75, 77, 79	$x_3+x_4+x_6+x_7+x_9+x_{11}+x_{17}+x_{18}+x_{20}+x_{22}+x_{26}+x_{28}+x_{29}+x_{31}+x_{34}+x_{35}+x_{37}+x_{38}+x_{39}+x_{41}+x_{46}+x_{50}+x_{51}+x_{54}+x_{55}+x_{56}+x_{58}+x_{61}+x_{65}+x_{66}+x_{67}+x_{74}+x_{76}+x_{78}+x_{79}+1$	791
1, 3, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 33, 36, 38, 40, 41, 43, 45, 47, 49, 53, 54, 56, 58, 62, 63, 69, 71, 74, 75, 77, 78, 80	$x_{43} + x_{47} + x_{58} + x_{70} + x_{74} + 1$	791

1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 38, 40, 41, 43, 45, 47, 49, 53, 56, 58, 62, 63, 69, 71, 75, 77, 80	$x_{12} + x_{17} + x_{26} + x_{27} + x_{29} + x_{30} + x_{32} + x_{40} + x_{43} + x_{45} + x_{46} + x_{49} + x_{53} + x_{54} + x_{56} + x_{59} + x_{62} + x_{64} + x_{65} + x_{67} + x_{69} + x_{72} + x_{74} + x_{75}$	792
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 41, 43, 45, 47, 49, 53, 54, 56, 58, 63, 69, 71, 75, 77, 78, 79, 80	$x_{12} + x_{26} + x_{39} + x_{56} + x_{68} + 1$	792
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 41, 43, 45, 47, 49, 53, 56, 58, 61, 63, 69, 71, 75, 77, 78, 79, 80	$x_{12}+x_{14}+x_{26}+x_{30}+x_{40}+x_{41}+x_{47}+x_{48}+x_{56}+x_{66}+x_{67}+x_{68}+x_{74}+x_{75}+1$	792
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 41, 43, 45, 47, 49, 53, 54, 56, 58, 63, 69, 71, 74, 75, 77, 78, 79, 80	$x_{16} + x_{43} + x_{56}$	792
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 61, 63, 65, 69, 71, 75, 77, 78, 79, 80	$x_{14} + x_{16} + x_{26} + x_{29} + x_{30} + x_{41} + x_{45} + x_{55} + x_{56} + x_{59} + x_{62} + x_{64} + x_{66} + x_{68} + x_{70} + x_{71} + x_{72} + 1$	792
1, 3, 6, 7, 8, 10, 12, 14, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 41, 43, 45, 47, 49, 53, 54, 56, 58, 61, 63, 69, 71, 75, 77, 80	$x_{45} + x_{72}$	793
1, 6, 7, 8, 10, 12, 14, 16, 19, 21, 23, 25, 27, 30, 31, 33, 38, 40, 43, 45, 47, 49, 51, 52, 56, 58, 63, 67, 69, 71, 73, 77, 80	$x_{10} + x_{55}$	798
1, 3, 5, 6, 7, 8, 10, 12, 15, 16, 19, 21, 23, 25, 27, 33, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 62, 63, 65, 69, 71, 74, 75, 77, 80	$x_{36} + x_{52} + x_{60} + x_{63}$	798
1, 3, 5, 6, 7, 8, 10, 12, 15, 16, 19, 21, 23, 25, 27, 33, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 62, 63, 65, 69, 71, 74, 75, 77, 80	$x_{10} + x_{17} + x_{27} + x_{36} + x_{37} + x_{40} + x_{52} + x_{59} + x_{60} + x_{63} + x_{66} + x_{67}$	798
1, 3, 5, 6, 7, 8, 10, 12, 14, 16, 19, 21, 23, 25, 33, 36, 38, 40, 43, 45, 47, 49, 53, 56, 58, 61, 62, 63, 65, 69, 71, 74, 75, 77, 78, 79	$x_{27} + x_{54} + x_{60}$	798
1, 3, 5, 6, 7, 8, 10, 12, 15, 16, 19, 21, 23, 25, 27, 33, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 61, 62, 63, 65, 69, 71, 74, 75, 77, 80	$x_6+x_{11}+x_{25}+x_{33}+x_{36}+x_{53}+x_{60}+x_{62}+x_{63}+x_{64}+x_{79}$	798
1, 3, 5, 6, 7, 8, 10, 12, 15, 16, 19, 21, 23, 25, 27, 33, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 61, 62, 63, 65, 69, 71, 74, 75, 77, 80	$x_6+x_{11}+x_{25}+x_{33}+x_{36}+x_{52}+x_{53}+x_{60}+x_{62}+x_{63}+x_{64}+x_{79}$	798
5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 34, 36, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 60, 61, 62, 63, 67, 69, 71, 74, 80	$x_{65} + x_{66} + x_{67} + 1$	798
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 43, 45, 47, 49, 54, 56, 58, 62, 69, 71, 73, 80	$x_{25} + 1$	799
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 40, 43, 45, 47, 49, 53, 56, 58, 63, 69, 71, 75, 77, 80	$x_{12} + x_{38} + x_{39} + x_{40}$	799

Trivium-799

maxterm bits	superpoly	round
1, 6, 8, 10, 12, 14, 18, 20, 23, 25, 27, 31, 33, 36, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 62, 63, 69, 73, 75, 77, 80	x_{60}	799
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 43, 45, 47, 49, 54, 56, 58, 62, 69, 71, 73, 80	$x_{25} + 1$	799
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 40, 43, 45, 47, 49, 53, 56, 58, 63, 69, 71, 75, 77, 80	$x_{25} + x_{40}$	799
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 40, 43, 45, 47, 49, 53, 56, 58, 63, 69, 71, 75, 77, 80	$x_{12} + x_{38} + x_{39} + x_{40}$	799
1, 5, 6, 7, 8, 10, 12, 14, 15, 16, 18, 20, 23, 25, 27, 33, 36, 38, 40, 41, 43, 47, 49, 53, 56, 58, 63, 69, 71, 75, 77, 80	$x_{67} + 1$	799
1, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 31, 33, 36, 38, 40, 41, 45, 47, 49, 53, 56, 58, 63, 69, 71, 75, 80	x_{42}	799
1, 5, 6, 7, 8, 10, 12, 14, 15, 19, 21, 23, 25, 27, 31, 36, 38, 40, 41, 45, 47, 49, 53, 56, 58, 63, 69, 71, 73, 75, 77, 80	x_{53}	799
1, 5, 6, 8, 10, 12, 14, 15, 16, 18, 19, 20, 21, 23, 25, 27, 31, 33, 36, 38, 40, 41, 45, 49, 54, 56, 62, 69, 73, 75, 77, 80	x_{64}	799
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 62, 63, 69, 71, 80	$x_{36} + 1$	799
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 40, 41, 43, 45, 47, 49, 53, 56, 58, 63, 65, 69, 71, 75, 77, 80	x_{38}	799
1, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 43, 45, 47, 54, 56, 58, 60, 62, 65, 69, 70, 71, 73, 80	x_{56}	799
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 33, 34, 36, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 62, 63, 69, 71, 74, 80	$x_{69} + 1$	799
1, 6, 7, 8, 10, 12, 14, 16, 19, 21, 25, 27, 30, 31, 33, 36, 38, 40, 41, 43, 45, 47, 49, 51, 52, 56, 58, 63, 67, 69, 71, 73, 77, 80	$x_{66} + 1$	799
1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 25, 27, 33, 34, 36, 38, 40, 43, 45, 47, 49, 53, 54, 56, 58, 62, 63, 67, 69, 71, 74, 80	x_{58}	799
1, 5, 6, 7, 8, 10, 12, 14, 15, 16, 19, 21, 23, 25, 27, 33, 36, 38, 40, 43, 45, 47, 49, 53, 54, 55, 56, 58, 62, 63, 67, 69, 71, 74, 80	x_{37}	799

Trivium-800

maxterm bits	superpoly	round
0, 5, 6, 7, 9, 11, 13, 17, 19, 20, 22, 24, 26, 30, 32, 33, 35, 37, 39, 42, 44, 46, 48, 52, 55, 57, 61, 62, 66, 68, 72, 74, 76, 79	x_{63}	800

References

1. Agnesse, A., Pedicini, M.: Cube attack in finite fields of higher order. In: Proceedings of 9th Australasian Information Security Conference, AISC 2011, pp. 9–14. ACS, Inc. (2011)
2. Agostini, E.: Bitlocker dictionary attack using GPUs. In: University of Cambridge Passwords 2015 Conference (2015). https://www.cl.cam.ac.uk/events/passwords2015/preproceedings.pdf
3. Ahmadian, Z., Rasoolzadeh, S., Salmasizadeh, M., Aref, M.R.: Automated dynamic cube attack on block ciphers: cryptanalysis of SIMON and KATAN. IACR Cryptology ePrint Archive **2015**, 40 (2015)
4. 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). doi:10.1007/978-3-642-03317-9_1
5. Baksi, A., Maitra, S., Sarkar, S.: New distinguishers for reduced round trivium and trivia-SC using cube testers. In: WCC2015-9th International Workshop on Coding and Cryptography 2015 (2015)
6. Bernstein, D.J.: Why haven't cube attacks broken anything? https://cr.yp.to/cubeattacks.html. Accessed 11 Nov 2016
7. Blum, M., Luby, M., Rubinfeld, R.: Self-testing/correcting with applications to numerical problems. In: ACM Symposium on Theory of Computing, pp. 73–83. ACM (1990)
8. 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). doi:10.1007/11836810_13
9. De Canniere, C., Preneel, B.: Trivium-specifications. eSTREAM, ECRYPT stream cipher project, report 2005/030 (2005)
10. 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). doi:10.1007/978-3-642-01001-9_16
11. 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). doi:10.1007/978-3-642-21702-9_10
12. Dinur, I., Shamir, A.: Applying cube attacks to stream ciphers in realistic scenarios. Cryptogr. Commun. **4**(3–4), 217–232 (2012)
13. Fan, X., Gong, G.: On the security of Hummingbird-2 against side channel cube attacks. In: Armknecht, F., Lucks, S. (eds.) WEWoRC 2011. LNCS, vol. 7242, pp. 18–29. Springer, Heidelberg (2012). doi:10.1007/978-3-642-34159-5_2
14. Fouque, P.-A., Vannet, T.: Improving key recovery to 784 and 799 rounds of trivium using optimized cube attacks. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 502–517. Springer, Heidelberg (2014). doi:10.1007/978-3-662-43933-3_26
15. Kleinjung, T., Lenstra, A.K., Page, D., Smart, N.P.: Using the cloud to determine key strengths. In: Galbraith, S., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 17–39. Springer, Heidelberg (2012). doi:10.1007/978-3-642-34931-7_3
16. Marks, M., Jantura, J., Niewiadomska-Szynkiewicz, E., Strzelczyk, P., Góźdź, K.: Heterogeneous GPU&CPU cluster for high performance computing in cryptography. Comput. Sci. **13**(2), 63–79 (2012)

17. Milo, F., Bernaschi, M., Bisson, M.: A fast, GPU based, dictionary attack to OpenPGP secret keyrings. J. Syst. Softw. **84**(12), 2088–2096 (2011)
18. O'Neil, S.: Algebraic structure defectoscopy (2007). Tools for Cryptanalysis 2007 Workshop. http://eprint.iacr.org/2007/378
19. Quedenfeld, F.M., Wolf, C.: Algebraic properties of the cube attack. IACR Cryptology ePrint Archive **2013**, 800 (2013)
20. Samorodnitsky, A.: Low-degree tests at large distances. In: Proceedings of 39th ACM symposium on Theory of Computing, pp. 506–515. ACM (2007)
21. Samorodnitsky, A., Trevisan, L.: A PCP characterization of NP with optimal amortized query complexity. In: Proceedings ACM Symposium on ToC, pp. 191–199. ACM (2000)
22. Shanmugam, D., Annadurai, S.: Secure implementation of stream cipher: trivium. In: Bica, I., Naccache, D., Simion, E. (eds.) SECITC 2015. LNCS, vol. 9522, pp. 253–266. Springer, Cham (2015). doi:10.1007/978-3-319-27179-8_18
23. Srinivasan, C., Pillai, U.U., Lakshmy, K., Sethumadhavan, M.: Cube attack on stream ciphers using a modified linearity test. J. Discret. Math. Sci. Cryptogr. **18**(3), 301–311 (2015)
24. Vielhaber, M.: Breaking ONE.FIVIUM by AIDA an algebraic IV differential attack (2007). http://eprint.iacr.org/2007/413
25. Winter, R., Salagean, A., Phan, R.C.-W.: Comparison of cube attacks over different vector spaces. In: Groth, J. (ed.) IMACC 2015. LNCS, vol. 9496, pp. 225–238. Springer, Cham (2015). doi:10.1007/978-3-319-27239-9_14
26. Zhang, S., Chen, G., Li, J.: Cube attack on reduced-round Quavium. ICMII-15 Adv. Comput. Sci. Res. (2015). doi:10.2991/icmii-15.2015.25