Chapter 6

Cube Attack on Stream Ciphers

6.1 Introduction

Most of the stream ciphers specified recently use two inputs to generate the keystream sequence: the secret variables representing the key, and the public variables representing the Initial Vector (IV). This gives a clear practical advantage over previously proposed stream ciphers with single input key. On the other hand, the use of an IV has considerable impacts on the cryptanalysis and on the formalization of the security requirements on stream ciphers. An adversary can compare the keystream sequences associated with several known, related or chosen IV values, and potentially derive information corresponding to the internal state values that could not be derived from one single keystream sequence. We assume that the attacker can simulate the cipher during the preprocessing phase, and can apply a chosen IV attack during the online phase. From the security perspective, several stream ciphers and block ciphers are found to be vulnerable to cryptanalysis due to their weak design criteria causing poor diffusion [4] in the cipher.

A cipher can be seen as a set of Boolean functions, each one mapping the key and IV bits to one of the keystream bits (plaintext and ciphertext bits for the case of block ciphers). Each of these functions can be represented Algebraic Normal Form (ANF). An algebraic attack is one which focuses on this representation, often to obtain and solve simultaneous linear equations in key bits. Algebraic attacks [30] have proved to be a powerful class of attacks which might threaten both block and stream ciphers. The idea is to set up an algebraic system of equations in key bits and try to solve. This kind of approach can be quite effective on stream ciphers which consist of a linear pseudorandom generator with non-linear combining functions acting on the outputs of the generator to produce the final output. It becomes very efficient if the system is of low degree. This kind of attack will work when certain Boolean functions used in the ciphering process have either low degree annihilators [100] or low degree multiples. The success of the attack depends on the keystream available and the effort required for solving the system of multivariate equations. In this chapter we focus on a special case of algebraic attacks called the cube attacks, which is effective against almost any cipher where atleast one of its output in polynomial representation has low degree. For cube attacks we need only a black-box access to the cipher, whereby we can assign various values to the IV (plaintext) bits and obtain the output bits. Both cube attack and higher-order differential cryptanalysis exploits this kind of weakness by taking advantage of low-degree polynomials. Since it is unlikely that a random polynomial of high degree will have low degree superpolys, some cryptographers believe that the cube attacks are applicable for systems of small degree only.

6.2 Cube Attack on Stream Ciphers

Cube attack was introduced in Eurocrypt 2009 by Itai Dinur and Adi Shamir [101] [102] as a chosen IV attack on symmetric primitives. This method has been further developed in [103] [104], where the authors introduced cube testers. In contrast to the cube attacks, cube testers can be used to mount distinguishers or to simply detect non-randomness in cryptographic algorithms and they do not require large pre-computations. A cipher is vulnerable if an output bit can be represented as a sufficiently low degree polynomial in input bits over the field $\mathbb{F}_2$. Cube attack exploits this property of the cipher and they can be applied even when this polynomial is unknown. In most of the cryptographic schemes, each keystream (ciphertext in the case of block ciphers) bit can be described by a multivariate polynomial $p(k_1, k_2, ..., k_n, v_1, v_2, ..., v_m)$ over $\mathbb{F}_2$ of secret variables $k_1, k_2, ..., k_n$ and public variables $v_1, v_2, ..., v_m$ which are plaintext bits in block ciphers or IV bits in stream ciphers. According to Dinur and Shamir, a cryptanalyst is allowed to tweak the master polynomial by assigning chosen values for the public variables and his goal is to solve the resultant system of polynomial equations in terms of their common secret variables. Cube attack is an extension of higher order differential attack and Michael Vielhabers Algebraic IV Differential Attack (AIDA) [105]. Dinur and Shamir recognize AIDA as a precursor of the cube attack which generalizes and improves AIDA. Vielhaber applied it on a Trivium variant with reduced number of initialization rounds (576 instead of 1152), providing linear equations in secret key bits in terms of sums of output bits which could be used for attacking the cipher. But no systematic method for obtaining such equations was described. One can also interpret this sum over the derived polynomials obtained from the master polynomial by assigning all the possible values to a subset of IV (or plaintext) variables, which is equivalent to differentiating the master polynomial with respect to these variables. The preprocessing phase is not key dependant and is performed once for a cryptosystem. In this phase, the attacker computes the coefficients of the secret variables in the linear superpoly. The main challenge of the attacker in the preprocessing phase is to find sufficiently many linearly independent superpolys. Given a multivariate polynomial $p(k_1, k_2, ..., k_n, v_1, v_2, ..., v_m)$ over $\mathbb{F}_2$ in algebraic normal form (ANF), and a term c_I containing variables from an index subset I. The polynomial p can be written as

$$p(k_1, k_2, ..., k_n, v_1, v_2, ..., v_m) = c_I \; p_I + q(k_1, k_2, ..., k_n, v_1, v_2, ..., v_m).$$

where p_I is called the superpoly of I in p, and each term in the polynomial q does not contain at least one variable from I. In the online phase, the n secret variables are set to unknown values $k_1, k_2, ..., k_n$ and the attacker is allowed to set the values of the m public variables $v_1, v_2, ..., v_m$ to any desired values and evaluate the master polynomial p. Finding linear superpolys in the preprocessing phase is a difficult task, but once they are found for a particular cryptosystem, then we can repeatedly use them to

easily find any secret key by solving the resultant system of linear superpolys. During the preprocessing phase, we need to test the linearity of the superpoly. The linearity testing problem is to check whether a function is close to linear by asking oracle queries to the function. This is the BLR (Blum-Luby-Rubinfeld) test, which has a time complexity of $\mathcal{O}(2^{2k+c})$ cipher operations, where k is the length of the key and c is the size of the cube. In section 6.3 we present a method for testing a superpoly for linearity with a time complexity of $\mathcal{O}(2^{c+1}(k^2+k))$ which we believe to be a sufficient condition. In section 6.4 we show few sparse linear superpolys of Trivium reduced to 576 number of rounds.
There have been various attempts to attack reduced variants of Trivium using cube attacks. Vielhaber in the year 2007 managed to recover 47 bits of the key after 576 rounds. Later Dinur and Shamir [101] [103] described a full key recovery in less than 2^{30} queries to Trivium reduced to 735 rounds and also recovered 35 key bits after 767 rounds in about 2^{36} queries. Recently, Fouque and Vannet [106] were able to find linear superpolys up to 784 rounds, which leads to an attack requiring 2^{39} queries. Using quadratic superpoly, they were also able to provide another attack up to 799 rounds with complexity 2^{40} queries.

A more general framework using probabilistic neutral key bits

Make partition of the initial vector IV $= (U, W)$ with U as input to f. Focus on single coefficient C of $f(U)$, example of this is a maximum degree monomial, which defines the cube attack [101] [105]. Hence cube attack can be a particular case of this analysis. The coefficient $C = C(K, W)$ depends on the key and remaining IV bits. The values of this Boolean function C are derived with the help of an oracle.

If mixing is not complete the following scenarios of attack is considered:

- Imbalanced structure of C can be used for distinguishing attacks [107] [108] [109].
- Sometimes, $C(K, W)$ for fixed W does not involve all key bits: like key recovery attack with reduced 576 rounds of Trivium [105].

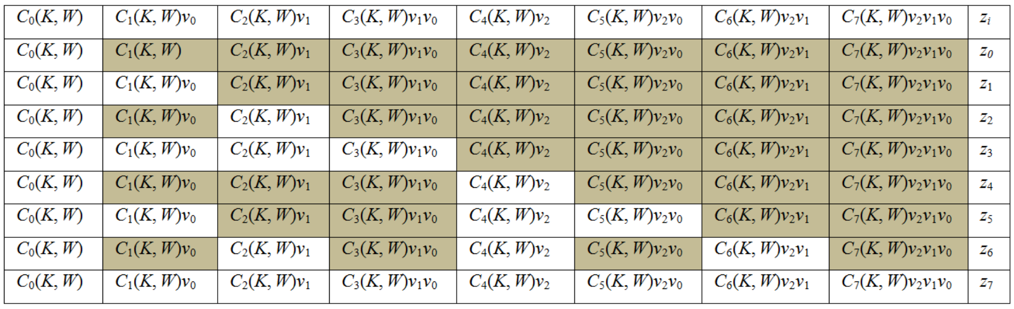
- More generally some key bits in $C(K, W)$ may have only a limited influence, leading to a key recovery attack [110].

Description of the Attack

Given $C(K, W)$, find approximation A which depends on subkey of only $t < n$ key bits, which reduces the search space from 2^n to 2^t [110]. Now partition the key bits K as $K = (L, M)$ with significant key bits L containing t bits and the non-signicant key bits M containing $n - t$ bits. Define $A(L, W)$ by replacing non-significant key bits in $C(K, W)$ with fixed values. Significant key bits L are obtained by choosing those key bits k_i whose $|\gamma_i| <$ threshold. Where γ_i is the neutrality measure corresponding to the key bit k_i with respect to $C(K, W)$ defined as the probability that complementing the key bit k_i does not change the output of $C(K, W)$. Then one has to guess and determine possible subkeys L and distinguish correct guess L from incorrect ones.
Compute $C(K, W)$ through oracle, for unknown key and say N randomly chosen values of W. For each choice of L do the following

- Compute $A(L, W)$ for the same N values of W.
- Check if $C(K, W)$ is equal to $A(L, W)$ for most of the N samples (optimal distinguisher).

Given a candidate subkey L, the entire key is veried by exhaustive search over remaining key part M. Consider an example $U = v_0, v_1, v_2$, where z_i is the keystream bit.



$C_0(K, W)$	$C_1(K, W)v_0$	$C_2(K, W)v_1$	$C_3(K, W)v_1v_0$	$C_4(K, W)v_2$	$C_5(K, W)v_2v_0$	$C_6(K, W)v_2v_1$	$C_7(K, W)v_2v_1v_0$	z_i
$C_0(K, W)$	▨ $C_1(K, W)$	▨ $C_2(K, W)v_1$	▨ $C_3(K, W)v_1v_0$	▨ $C_4(K, W)v_2$	▨ $C_5(K, W)v_2v_0$	▨ $C_6(K, W)v_2v_1$	▨ $C_7(K, W)v_2v_1v_0$	z_0
$C_0(K, W)$	$C_1(K, W)v_0$	▨ $C_2(K, W)v_1$	▨ $C_3(K, W)v_1v_0$	▨ $C_4(K, W)v_2$	▨ $C_5(K, W)v_2v_0$	▨ $C_6(K, W)v_2v_1$	▨ $C_7(K, W)v_2v_1v_0$	z_1
$C_0(K, W)$	▨ $C_1(K, W)v_0$	$C_2(K, W)v_1$	▨ $C_3(K, W)v_1v_0$	▨ $C_4(K, W)v_2$	▨ $C_5(K, W)v_2v_0$	▨ $C_6(K, W)v_2v_1$	▨ $C_7(K, W)v_2v_1v_0$	z_2
$C_0(K, W)$	$C_1(K, W)v_0$	$C_2(K, W)v_1$	$C_3(K, W)v_1v_0$	▨ $C_4(K, W)v_2$	▨ $C_5(K, W)v_2v_0$	▨ $C_6(K, W)v_2v_1$	▨ $C_7(K, W)v_2v_1v_0$	z_3
$C_0(K, W)$	▨ $C_1(K, W)v_0$	▨ $C_2(K, W)v_1$	▨ $C_3(K, W)v_1v_0$	$C_4(K, W)v_2$	▨ $C_5(K, W)v_2v_0$	▨ $C_6(K, W)v_2v_1$	▨ $C_7(K, W)v_2v_1v_0$	z_4
$C_0(K, W)$	$C_1(K, W)v_0$	▨ $C_2(K, W)v_1$	▨ $C_3(K, W)v_1v_0$	$C_4(K, W)v_2$	$C_5(K, W)v_2v_0$	▨ $C_6(K, W)v_2v_1$	▨ $C_7(K, W)v_2v_1v_0$	z_5
$C_0(K, W)$	▨ $C_1(K, W)v_0$	$C_2(K, W)v_1$	▨ $C_3(K, W)v_1v_0$	$C_4(K, W)v_2$	▨ $C_5(K, W)v_2v_0$	$C_6(K, W)v_2v_1$	▨ $C_7(K, W)v_2v_1v_0$	z_6
$C_0(K, W)$	$C_1(K, W)v_0$	$C_2(K, W)v_1$	$C_3(K, W)v_1v_0$	$C_4(K, W)v_2$	$C_5(K, W)v_2v_0$	$C_6(K, W)v_2v_1$	$C_7(K, W)v_2v_1v_0$	z_7

Figure 6.1: Coefficients of 3-variable (v_0, v_1, v_2) function f

The shaded region in the figure 6.1 represents the coefficients $C_k(K, W)$ of f which does not contribute to generate either z_i or a linear combination of z_i $(i = 1, 2, ...7)$. Also from the above figure it is clear that cube attack [101] is simply evaluating $C_7(K, W) = \oplus_{i=0}^{7} z_i$, which is the highest degree coefficient of function *f*.

6.3 Linearity Tests

Traditionally, cryptographic applications designed on hardware have always tried to take advantage of the simplicity of functions over the field $\mathbb{F}_2$ to reduce costs and improve performance. We present some definitions and algebraic preliminaries required to show our result. A Boolean function in n variables is a function from $\mathbb{F}_2^n$ to $\mathbb{F}_2$, n being a positive integer, can be represented as

$$f(x_1, x_2, ..., x_n) = \sum_{u \in \mathbb{F}_2} \lambda_u (\prod_{i=1}^{n} {x_i}^{u_i}),\ \lambda_u \in \mathbb{F}_2, u = (u_1, u_2, ..., u_n), u_i \in \mathbb{F}_2$$

This representation of f is called the algebraic normal form of *f*. The algebraic degree of *f*, denoted by deg(f) is the maximal value of the Hamming weight of u such that $\lambda_u \neq 0$. If there exits atleast one monomial x^u of f with degree > 1 $(\lambda_u \neq 0)$, then the function f is said to be nonlinear. For simplicity we shall define one degree monomials as pendants. So there exist a maximum of n pendants for an n variable Boolean function.

Definition 6.3.1. A function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is linear if there exists a vector $a = (a_1, a_2, ..., a_n) \in \mathbb{F}_2^n$ such that $f(x) = \bigoplus_{i=1}^{n} a_i x_i$ (here $\bigoplus$ is the XOR operation).

Definition 6.3.2. A function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is linear if $f(x) \oplus f(y) = f(x \oplus y)$, $\forall\ x, y \in \mathbb{F}_2^n$.

The definitions 6.3.1 and 6.3.2 are equivalent.

6.3.1 BLR Linearity Test for Boolean Functions

The study of linearity testing was initiated by Blum, Luby and Rubinfeld [111]. They showed that if a function f satisfies the identity $f(x \oplus y) = f(x) \oplus f(y)$ for a large fraction of pairs $x, y \in \mathbb{F}_2^n$, with probability $1 - \epsilon$,

$\epsilon \in$[0, 1], then f is ϵ-close to being linear. This can be tested with $\mathcal{O}(\frac{1}{\epsilon})$ queries by simply verifying that the above identity holds for randomly selected pairs $x, y \in \mathbb{F}_2^n$. The exact query complexity of this problem has been studied in [112]. In cube attack the BLR linearity test is used probabilistically. Therefore the chances of detecting the presence of a high degree term are low, as a result of this a nonlinear polynomial may be detected as a linear polynomial in some cases.

In this chapter we define three filters (A, B, C) for testing the linearity of Boolean functions. A Boolean function is passed through these filters one after the other to check its linearity. It is necessary for any Boolean function to pass through these filters to prove its linearity and the sufficiency condition is yet to be proved.

6.3.2 Proposed Linearity Test for Boolean Functions

Let $e_i = (0, 0, ...0, 1, 0, ...0, 0)$, $1 \leq i \leq n$ be a binary string obtained by complementing the i-th bit of the all zero n bit string $e_0 = (0, 0, ...0)$. Let $e_{0j} = (1, 1, ..., 1, 0, 1, ..., 1, 1)$, $1 \leq j \leq n$ be an n bit binary string obtained by complementing the j-th bit of the all one n bit string $\hat{e}_0 = (1, 1, ...1)$. Let $e_{ij}, 1 \leq i, j \leq n, i \neq j$ be an n bit string obtained by complementing i-th and j-th bits in $\hat{e}_0$. And $\hat{e}_{ij}, 1 \leq i, j \leq n, i \neq j$ be an n bit string obtained by complementing i-th and j-th bits in e_0.

Filter-*A*

Given a Boolean function f, this filter evaluates whether f is nonlinear or it has an even number of nonlinear monomials. If f is nonlinear then we need to stop the filtering process, otherwise pass f to the Filter-B given below.

a) Compute $f(e_i), 0 \leq i \leq n$.

b) Construct the pendant set
$P = \{i : f(e_0) \neq f(e_i), 1 \leq i \leq n\} = \{i_1, i_2, ..., i_r\}, r \leq n$.
$i \in P$, implies x_i is a pendant in the ANF of f.

c) If $f(\hat{e}_0) \neq f(e_P))$, then f is nonlinear, otherwise send f through Filter-B. Here e_P is an n bit string obtained by complementing the i_1-th, i_2-th,..., i_r-th bits of the all zero string e_0.

Filter-*B*

Given a Boolean function *f*, this filter evaluates whether f is nonlinear or it gives a list of variables which exits in odd number of monomials of *f*. If the indices of these list of variables equals the pendant set P, then there is still an ambiguity in accepting f to be linear and in which case we send f through Filter-C.

a) Compute $f(e_{0j}), 1 \leq j \leq n$.

b) Construct the odd count set $O = \{j : f(\hat{e}_0) \neq f(e_{0j}), 1 \leq j \leq n\}$. Here $j \in O$ implies x_j exits in odd number of monomials in the ANF of *f*. One can rewrite the function f as $f(x) = x_j q(x) + r(x)$, where the functions $q(x)$ and $r(x)$ does not contain the variable x_j. Then $f(e_{0j}) = r(\hat{e}_0)$ and $f(\hat{e}_{0j}) = q(\hat{e}_0) + r(\hat{e}_0)$. If the variable x_j exists in even number of monomials of f then $q(\hat{e}_0) = 0$ which implies $f(e_{0j}) = f(\hat{e}_0)$ and if x_j exists odd number of times then $q(\hat{e}_0) \neq 0$.

c) If $O \neq P$ then f is said to be nonlinear and then stop the filtering process. Otherwise f is send through Filter-C to prove its linearity.

Filter-*C*

Given a Boolean function f and O, this filter evaluates whether f is linear or not.

a) Compute the sets

$$O_i = \{j : f(e_{0i}) \neq f(e_{ij}), 1 \leq j(\neq i) \leq n\}, i = 1, 2, ..., n$$

$$\hat{O}_i = \{j : f(e_i) \neq f(\hat{e}_{ij}), 1 \leq j(\neq i) \leq n\}, i = 1, 2, ..., n$$

b) If these sets O_i and $\hat{O}_i, 1 \leq i \leq n$ satisfy the following condition, then the function f is linear, otherwise nonlinear

$$O_i = \hat{O}_i = \begin{cases} P - \{i\} & \text{if } i \in P \\ P & otherwise \end{cases}$$

6.3.3 Pseudocode of the Proposed Algorithm

INPUT: An n variable Boolean function f
OUTPUT: Either f is linear or nonlinear

```
1. Compute $P = \{i_1, i_2, ..., i_r\} = \{i : f(e_0) \neq f(e_i), 1 \leq i \leq n\}$.
2. If $f(\hat{e}_0) \neq f(e_P)$ then print $f$ is nonlinear and go to step 7.
3. Compute $O = \{j : f(\hat{e}_0) \neq f(e_{0j}), 1 \leq j \leq n\}$.
4. If $P \neq O$ then print $f$ is nonlinear and go to step 7.
5. Compute $O_i = \{j : f(e_{0i}) \neq f(e_{ij}), 1 \leq j(\neq i) \leq n\}, i = 1, 2, ..., n$,
   and
   $\hat{O}_i = \{j : f(e_i) \neq f(\hat{e}_{ij}), 1 \leq j(\neq i) \leq n\}, i = 1, 2, ..., n$
6. If
       $O_i = \hat{O}_i = \begin{cases} P - \{i\} & \text{if } i \in P \\ P & otherwise \end{cases}$
   then print $f$ is linear else print $f$ is nonlinear.
7. End the algorithm.
```

6.4 Cube Attack on Trivium with Reduced Rounds

Trivium [3] is a synchronous stream cipher designed to provide a flexible trade-off between speed and gate count in hardware, and reasonably efficient software implementation. It consists of a 288 stage nonlinear feedback shift register with 3 nonlinear feedback functions. The initialization phase consists of 4×288 cycles without revealing the output. We consider a variant of Trivium for our analysis and perform cube attack on it. The variant considered differ from Trivium only in the number of initialization rounds after which the system generates output. The number of initialization rounds is 576 instead of 1152 rounds for Trivium. In the preprocessing stage we found extremely sparse linear expressions involving unknown key bits. These expressions were obtained by summing up the derived polynomials over cubes. The dimensions of the cubes were chosen heuristically. In obtaining all the linear superpolys given below, the public variables (IV variables) except the cube variables are all set to zero. The following table shows the linear expressions in unknown key bits corresponding to the cubes given in the first column. The second column describes the output bit index after key/IV initialization for 576 rounds. In the table given below the output bit index O/P bit $= r$ implies the output bit corresponding to $(576 + r)$-th round.

From table 6.1, we get 69 extremely sparse linearly independent linear equations obtained for smaller cubes for Trivium cipher reduced to 576 rounds.

6.5 Conclusion

In cube attack the BLR linearity test is used probabilistically to test the linearity of the superpoly. The linearity testing problem is to check whether a function is close to linear by requesting oracle queries to the function. Therefore the chances of detecting the presence of a high degree term is low, as a result a nonlinear polynomial may be detected as a linear polynomial in some cases. In this chapter we give a modified linearity test over $\mathbb{F}_2$ which detects linear polynomials with high probability. The chapter contributes to the analysis of 576 round Trivium using cube attack. This analysis provides a different set of sparse linear equations, compared to that of the similar work carried out by Vielhaber in 2007. The task for retrieving linear equations for the full round of the cipher still remains open and is yet to be investigated.

Table 6.1: Linear superpolys for Trivium with 576 initialization rounds

3,11,20,30,41,65	2	x_0	10,21,26,30,48,67	14	x_{32}
4,7,12,15,27,66	1	x_1	4,7,12,15,37,46,76	6	x_{33}
3,9,15,18,30,65	2	$x_2 + x_{65}$	5,9,16,19,30,41	7	x_{34}
4,11,16,19,26,46	2	x_3	4,26,48,50,58,68,70	6	x_{35}
4,7,12,15,21,78	1	x_4	3,32,46,50,57,65,73	4	x_{36}
14,18,28,46,52,61	2	x_5	9,35,41,48,53,58,70	2	x_{37}
4,7,11,28,35,55	20	x_6	10,36,42,49,53,56,74	3	x_{38}
3,7,14,18,23,63	35	x_7	13,31,50,55,58,63,69	1	x_{39}
4,7,12,15,23,54	11	x_8	13,36,41,48,54,55,73	19	x_{40}
5,8,15,19,23,67	12	x_9	13,15,23,47,52,63	0	x_{41}
6,11,15,19,26,65	18	x_{10}	8,38,45,52,56,68	4	$x_{42} + x_{57}$
1,11,22,30,40,52	8	x_{11}	1,10,22,29,39,66	3	x_{43}
14,18,31,43,53,69	8	$x_{12} + 1$	1,9,15,20,36,42,51,60	44	x_{44}
13,15,28,45,53,67	8	x_{13}	2,11,24,32,43,52	9	x_{45}
10,17,27,29,37,45	12	x_{14}	20,29,38,46,57,78	12	x_{46}
1,13,18,45,71,78	6	x_{15}	4,11,20,28,40,45,57	11	x_{47}
3,10,12,18,27,69	5	x_{16}	20,41,44,51,54,61,64,68	23	x_{48}
13,20,27,31,35,45	3	x_{17}	0,7,12,24,36,45	11	x_{49}
14,16,33,49,52,64	9	x_{18}	17,35,52,55,61,64	9	x_{50}
4,7,12,15,30,73	2	x_{19}	7,14,49,64,71,76,79	9	x_{51}
1,13,18,45,71,78	20	x_{20}	10,16,25,45,50,69	6	x_{52}
16,29,32,49,53,75	1	x_{21}	0,11,21,31,39,47,51	39	x_{53}
3,9,18,29,38,43	20	x_{22}	0,15,52,56,60,74	1	$x_{54} + 1$
0,9,15,19,26,70	5	x_{23}	16,44,63,68,75,79	1	x_{55}
15,20,27,38,42,74	6	x_{24}	4,10,15,19,29,37	16	x_{56}
4,6,19,21,35,65	5	$x_{25} + 1$	7,19,53,66,68,78	11	x_{57}
4,11,16,19,24,42	1	x_{26}	0,10,18,24,69,77	16	x_{58}
5,9,16,19,23,30,44	13	x_{27}	1,7,16,21,26,76	23	x_{59}
10,16,26,29,41,44	31	x_{28}	4,7,12,15,43,73	11	x_{60}
10,20,27,29,37,45	12	x_{29}	6,21,25,32,44,72	5	x_{61}
17,23,26,34,45,61	11	x_{30}	4,13,14,25,72,75	10	x_{62}
1,21,27,35,44,52	7	x_{31}	21,41,62,68,74,77	9	x_{63}
4,9,19,22,38,62	4	$x_{66} + 1$	4,41,63,68,75,77	9	x_{64}
0,13,16,49,72,78	8	x_{67}	0,10,13,18,30,60	6	x_{65}
4,7,12,15,23,79	7	x_{68}			