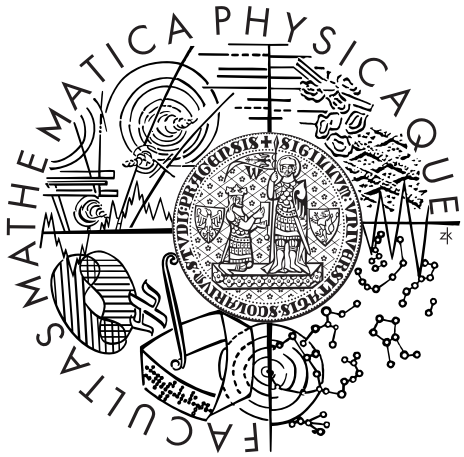
Charles University in Prague
Faculty of Mathematics and Physics

BACHELOR THESIS



Josef Bárta

Cube Attacks

Department of Algebra

Supervisor of the bachelor thesis:	RNDr. Michal Hojsík, Ph.D.
Study programme:	Mathematics
Specialization:	Mathematical Methods of Information Security

Prague 2014

I would like to thank my supervisor RNDr. Michal Hojsík, Ph.D. for his advice, ideas and, most of all, the time he spent proof-reading and consulting with me and moral support he gave me. I also thank my parents for providing me the background to work on the thesis.

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In ..... on ..... Author signature

Název práce: Krychlové útoky

Autor: Josef Bárta

Katedra: Katedra algebry

Vedoucí bakalářské práce: RNDr. Michal Hojsík, Ph.D., Katedra algebry

Abstrakt: Na základě Kubického útoku od Itai Dinura a Adi Shamira a další, ve své podstatě podobné, metody jsme navrhli novou techniku linearizace polynomů, která se ukázala být silnější, než samotný Kubický útok. Navíc uvádíme detailní popis a formální důkaz nejen našich nálezů, nýbrž i Kubického útoku. Nakonec demonstrujeme výsledky našeho snažení na Triviu se zkráceným klíčem a inicializačním vektorem. Čistě technikami popsanými v této práci jsme dokázali linearizovat polynom reprezentující bit keystreamu po 621 inicializačních iteracích kryptosystému, v porovnání s 581 inicializačními iteracemi původní techniky.

Klíčová slova: krychlové útoky, kryptoanalýza, proudové šifry

Title: Cube Attacks

Author: Josef Bárta

Department: Department of Algebra

Supervisor of the bachelor thesis: RNDr. Michal Hojsík, Ph.D., Department of Algebra

Abstract: Based on the Cube Attack by Itai Dinur and Adi Shamir and another, in the essence similar, method we devised a new polynomial linearisation technique, which proved to be more powerful, than the Cube Attack alone. Moreover, we present detailed description with formal proof not only of our findings, but also of the Cube Attack. Finally, we demonstrate the results of our efforts on a Trivium variant that is reduced in key and initialisation vector bit count. We managed to linearise polynomials representing a keystream bit output after up to 621 initialisation rounds using purely techniques described in this thesis, compared to 581 initialisation rounds with original attack.

Keywords: cube attacks, cryptanalysis, stream ciphers

Contents

Chapter 1

Introduction

In 2008 Adi Shamir et. al. [3] presented a new attack on the Trivium cryptosystem with reduced number of initialisation rounds. It is called the Cube Attack and is based on linearising of polynomials expressing the keystream bits, where the key and initial vector bits are the variables. The technique presented in the paper was aiming to recover a linear polynomial from a polynomial, the explicit representation of which is unkown. When attacking the full Trivium cryptosystem only with reduced number of initialisation rounds, this is an absolutely reasonable approach, since building the explicit representation of such polynomials is not feasible by currently available technology. Moreover, it would be easier to recover the key by brute force, since the polynomials in the cryptosystem after a higher number of initialisation rounds can be considered to be random, the memory complexity of expressing the keystream (and therefore naturally state) bits as polynomials in key and initial vector would be more than 2^{160}. The memory complexity of brute-force attack is negligible and time complexity would be 2^{80}. That is still enough to comply with what is demanded from a lightweight cryptosystem.

For this thesis we aimed to present a detailed description of the Cube Attack and then to devise a generalisation, which could help push the boundaries of usability of the Cube Attack. Other important target of ours are of course the polynomials as such. Therefore we decided to actually compute the polynomial expression of the state and keystream bits of Trivium reduced in the number of the bits used as variables and then to analyse them without having to do any guessing. More specifically, we wanted to assess, whether the polynomials are linearisable using the techniques devised by us and whether they are any more effective than the original Cube Attack.

In the next chapter there is described the basic notation we use throughout the thesis. Thereafter we "translate" the Cube Attack into our notation and we present its detailed description. In further sections we describe in the same manner another technique that can be used for attack in a similar way to the Cube Attack, which we, due to its problematic computability, simplify into two easier, but nonetheless effective techniques and a combination of both our method and the Cube Attack, one version of which proved to be more effective than the Cube Attack. The details about analysis of the polynomials and a description of the cryptosystem they represent can be found in the second to last chapter.

Chapter 2

TC-linearisation of tweakable polynomials

In this chapter we describe the theory behind the Cube Attack. Further in the text we define a technique, which is in itself very simple, but in its full variant could prove to be a very powerful way of linearising polynomials, if it was not for the computational complexity of the algorithm the equivalent condition yields. Nevertheless, we also present two simple variants, one of which proves in the next chapter to be quite powerful, when teamed up with the Cube Attack.

2.1 Tweakable polynomials

In this section, we introduce some notation and define the classes of polynomials we will be working with.

Throughout this theses, we denote by $[n]$ the set $\{0, 1, \ldots, n-1\}$ for any $n \in \mathbb{N}$.

Definition 2.1.1. A Boolean function in $n \in \mathbb{N}$ variables is a function of the form $f : \mathbb{F}_2^n \to \mathbb{F}_2$. The set of all Boolean functions in n variables is denoted by $\mathcal{B}_n$, i.e.

$$\mathcal{B}_n = \{f : \mathbb{F}_2^n \to \mathbb{F}_2\}.$$

Algebraic normal form (ANF) of a Boolean function f is its representation as a polynomial $f(x_0, \ldots, x_{n-1}) \in \mathbb{F}_2[x_0, \ldots, x_{n-1}]$ such that none of its monomials contain any variable in degree greater than one.

Theorem 2.1.2. *For each Boolean function, there exist a unique algebraic normal form.*

Proof of the theorem can be found in [1].

For $I \subseteq [n]$, we will use x_I to denote the monomial $\prod_{i \in I} x_i$. So for every Boolean function $f \in \mathcal{B}_n$ there exists a unique set $\mathcal{I} \subseteq \mathcal{P}([n])$ such that

$$f(x_0, \ldots, x_{n-1}) = \sum_{I \in \mathcal{I}} x_I.$$

We will write $x_I \in f$ if $I \in \mathcal{I}$.

Definition 2.1.3. Let $m, n \in \mathbb{N}$. We define set of secret variables $X = \{x_i; i \in [n]\}$ and set of public variables $Y = \{y_j; j \in [m]\}$.

Later on, the secret variables will represent the secret key, while the public variables will represent the initialisation vector of a stream cipher, which is public and can be potentially set by an attacker.

In the rest of the thesis we will use the ANF representation of Boolean functions and use the notation $\mathcal{B}[X]$, $\mathcal{B}[Y]$, $\mathcal{B}[X, Y]$ for Boolean functions (polynomials) in variables X, Y or $X \cup Y$ respectively.

Definition 2.1.4. We call a polynomial p tweakable, if $p \in \mathcal{B}[X, Y]$ and fully tweakable, if $p \in \mathcal{B}[Y]$.

2.2 Basic Cube Attack

This section describes the basic principles of the Cube Attack in the same way it was done in [3] to introduce the basic principles. For demonstration purposes we will not be distinguishing secret and public variables and thus we use fully tweakable polynomials. We also present a more formal proof of the main theorem from [3].

Definition 2.2.1. [3] Let $p \in \mathcal{B}[Y]$ be a polynomial and $J \subseteq [m]$ a variable index subset. A superpoly of J in p is a polynomial $p_{S(J)} \in B[Y]$ such that

$$p(Y) = y_J \cdot p_{S(J)}(Y) + q_J(Y) \tag{2.2.1}$$

where $q = \sum_{J \not\subseteq J'} b_{J'} y_{J'}$, $b_{J'} \in \mathbb{F}_2$. We call y_J a maxterm, if the superpoly $p_{S(J)}$ is a linear, non-constant polynomial.

Note. The superpoly $p_{S(J)}$ does not contain any variables indexed by J.

Example 2.2.2. Let $p \in \mathcal{B}[Y]$ be a polynomial

$$p = y_0y_3y_4y_5 + y_1y_3y_4y_5 + y_0y_3y_5 + y_0y_2y_3y_4 + y_0y_1 + y_1y_2y_4 + y_2y_3y_4 + y_3y_4 + y_3y_5 + y_3 + y_4 \ .$$

We can factor out the monomial $y_J = y_3y_4y_5$ so we get

$$p = \underbrace{y_3y_4y_5}_{y_J} \cdot \overbrace{(y_0 + y_1)}^{p_{S(J)}(Y)} + \underbrace{y_0y_3y_5 + y_0y_2y_3y_4 + y_0y_1 + y_1y_2y_4 + y_2y_3y_4 + y_3y_4 + y_3y_5 + y_3 + y_4}_{q_J(Y)}$$

In this case, y_J is a maxterm of J in p.

Definition 2.2.3. For an index subset $J \subseteq [m]$, $|J| = k$ we define a summation cube C_J as the set of k-tuples of variables $y_j : j \in J$ where all possible combinations of values of variables y_j are assigned. We can also understand C_J as a vector space $\mathbb{F}_2^k$ with information about indices of the variables. Hence we set $dim(C_J) = k$.

Definition 2.2.4. [3] For every polynomial $p \in \mathcal{B}[Y]$ and for any k-dimensional summation cube C_J, $J \subseteq [m]$ we define

$$p_J := \sum_{v \in C_J} p_{|v}$$

where $p_{|v}$ is a derived polynomial with $m - k$ variables $\{y_j : j \in [m] \setminus J\}$ and the variables indexed with J are assigned values from the k-tuple v.

Now we can present and prove a vital property of the superpoly of J in p, which is the main theorem in [3].

Proposition 2.2.5. *[3] For any polynomial $p \in \mathcal{B}[Y]$ and variable subset J, $p_J = p_{S(J)}$.*

Proof. First, we use the expression of the polynomial p from (2.2.1) and the fact, that the polynomial $p_{S(J)}$ does not contain variables with indices from J:

$$p_J = \sum_{v \in C_J} p_{|v} = \sum_{v \in C_J} (y_J p_{S(J)} + q)_{|v} = \sum_{v \in C_J} (y_{J|v} p_{S(J)} + q_{|v}) = \sum_{v \in C_J} y_{J|v} p_{S(J)} + \sum_{v \in C_J} q_{|v} \ .$$

Now we have to realize, that $p_{S(J)}$ will be added if and only if all the cube variables are all set to 1, so

$$p_J = p_{S(J)} + \sum_{v \in C_J} q_{|v} = p_{S(J)} + \sum_{v \in C_J} \Big(\sum_{J' \subseteq [m]} b_{J'} y_{J'} \Big)_{|v} = p_{S(J)} + \sum_{J' \subseteq [m]} \sum_{v \in C_J} (b_{J'} y_{J'})_{|v} \ .$$

For J' we have $J \not\subseteq J'$ and hence each $b_{J'} y_{J'}$ will be added up an even number of times and hence $\sum_{v \in C_J} (b_{J'} y_{J'})_{|v} = 0$ for all J'. This leaves us with $p_J = p_{S(J)}$. □

For our purposes, from now on, we shall call this technique of summing (partial) evaluations of a polynomial the **C-linearisation of fully tweakable polynomials**.

2.3 C-linearisation of tweakable polynomials

In this section we describe the C-linearisation (cube attack) on tweakable polynomials. We present a clear description of what makes a polynomial C-linearisable. In [3] this part was skipped, for they dealt with black box polynomials which demand a different approach than polynomials the explicit representation of which is known.

Definition 2.3.1. We call a polynomial $p \in \mathcal{B}[X, Y]$ C-linearisable, if there exists $J \subseteq [m]$ such that $p_J(X, Y = (1, ..., 1))$ is linear.

For purposes of C-linearisation we present the following grouping of monomials: Let $p \in \mathcal{B}[X, Y]$ be a tweakable polynomial. Then we can write

$$p = \sum_{(I,J)\in\mathcal{I}} x_I y_J = \sum_{l\in L_p} l + \sum_{b\in B_p} b + \sum_{h\in H_p} h$$

where $\mathcal{I} \subseteq \mathcal{P}([n]) \times \mathcal{P}([m])$ and

$$\begin{aligned} B_p &= \{x_I y_J \in p : |I| = 0\} \\ L_p &= \{x_I y_J \in p : |I| = 1\} \\ H_p &= \{x_I y_J \in p : |I| > 1\}. \end{aligned}$$

The set B_p contains all monomials consisting purely of public variables and the free monomial. L_p contains all monomials consisting of exactly one secret and any number and combination of public variables. H_p consists of monomials with two or more secret variables. We can plainly see that $L_p \cup B_p \cup H_p$ contains all monomials of p.

Before we present the condition which describes precisely a C-linearisable polynomial, we present a simple lemma about C-linearisability:

Lemma 2.3.2. *Let $p \in B[X,Y]$ be a tweakable polynomial. If $L_p = \emptyset$, then p is not C-linearisable.*

Proof. If there is no monomial that is linear in secret variables, there is definitely no monomial $y_J,\ J \subseteq [m]$, such that $p_{S(J)}$ is linear in secret variables. □

Now we can propose an equivalent definition of a C-linearisable tweakable polynomial:

Proposition 2.3.3. *A tweakable polynomial $p \in \mathcal{B}[X,Y]$ is C-linearisable if and only if*

$$\exists x_i y_J \in L_p,\ : (\forall y_{J'} x_I \in H_p : y_J \not| y_{J'})$$

Proof. We shall prove the first implication by contradiction, the second directly:
"⇒": For contradiction, we assume that p is C-linearisable and $\forall y_J x_i \in L_p \exists y_{J'} x_I \in H_p : y_J | y_{J'}$. This implies that for every choice of J will in the superpoly $p_{S(J)}$ remain a monomial that is non-linear in secret variables, i.e. the superpoly will contain $\frac{y_{J'}}{y_J} x_I$ and $|I| \geq 2$ as $y_{J'} x_I \in H_p$. Thus the contradiction.
"⇐": We assume that $\exists y_J x_i \in L_p\ \forall y_{J'} x_I \in H_p : y_J \not| y_{J'}$. That implies y_J is a maxterm, which yields a superpoly $p_{S(J)}$ that is linear in secret variables. □

This proposition yields a straightforward Algorithm 1 that checks if $p(X,Y)$ is C-linearisable.

Example 2.3.4. In this example we rewrite the polynomial from previous example into the notation of the tweakable polynomials with distinguished secret and public variables. We shall have $m = n = 3$. So let $p \in \mathcal{B}[X,Y]$ be a tweakable polynomial:

$$p = x_0y_0y_1y_2 + x_1y_0y_1y_2 + x_0y_0y_2 + x_0x_1y_0y_1 + x_1x_2y_1 + x_2y_0y_1 + y_0y_1 + y_0y_2 + y_0 + y_2$$

We can factor out $y_I = y_0y_1y_2$, so we obtain

$$p = y_0y_1y_2 \cdot (x_0 + x_1) + x_0y_0y_2 + x_0x_1y_0y_1 + x_1x_2y_1 + x_2y_0y_1 + y_0y_1 + y_0y_2 + y_0 + y_2$$

where $x_0 + x_1$ is the linear superpoly of $I = \{0,1,2\}$ in p and y_I is a maxterm.

Algorithm 1 C-linearisation

```
for x_i y_J ∈ L_p do
    C-linearisable = True
    for x_{I'} y_{J'} ∈ H_p do
        if J ⊆ J' then
            C-linearisable = False
            break
        end if
    end for
    if C-linearisable == True then return J
    end if
end for
return False
```

2.4 T-linearisation of tweakable polynomials

Now we present T-linearisation, a technique we devised and describe to aid the C-linearisation to be as effective as possible when linearising a polynomial.

Definition 2.4.1. We call a polynomial $p \in \mathcal{B}[X, Y]$ T-linearisable, if

$$\exists J \subseteq [m] : (\exists v \in C_J : p_{|v} \text{ is linear in secret variables})$$

In other words there exists a (partial) evaluation of the polynomial in public variables that results in $p_{|v}$ being linear in secret variables.

Proposition 2.4.2. *Let $p \in \mathcal{B}[X, Y]$. If*

$$\exists l \in L_p,\, l = x_i y_J :\ (\forall h \in H_p\ \exists j \in [m] \setminus J : y_j | h)$$

then p is T-linearisable. Specially, we say that p is T1-linearisable.

Proof. If there is such l that the condition holds, then every $h \in H_p$ can be eliminated by setting respective $y_j = 0$ while the secret part of l will be kept in the polynomial by setting the public variables indexed by J to one. □

As with C-linearisation, this proposition basically constructs a simple T1-linearisation algorithm Alg. 2.

We recall the polynomial from previous example to demonstrate the T1-linearisation:

Example 2.4.3. Let

$$p = x_0y_0y_1y_2 + x_1y_0y_1y_2 + x_0y_0y_2 + x_0x_1y_0y_1 + x_1x_2y_1 + x_2y_0y_1 + y_0y_1 + y_0y_2 + y_0 + y_2$$

We can linearise this polynomial in secret variables by setting $y_1 = 0$, which gives us a non-constant polynomial, that is linear in secret variables

Algorithm 2 T1-linearisation

```
for x_i y_J ∈ L_p do
    T1-linearisable = True
    for x_{I'} y_{J'} ∈ H_p do
        if J' \ J == ∅ then
            T1-linearisable = False
            break
        end if
    end for
    if T1-linearisable == True then return J
    end if
end for
return False
```

$$p_{|y_1=0} = x_0y_0y_2 + y_0y_2 + y_0 + y_2$$

and finally, by setting $y_0 = y_2 = 1$ we get

$$p_{|y_1=0,y_0=y_2=1} = x_0 + 1$$

which is a linear polynomial in secret variables only.

Corollary 2.4.4. *C-linearisability does not imply T1-linearisability.*

Proof. Consider polynomial $p = x_0y_0y_1 + x_0x_1y_1$. This polynomial is clearly not T1-linearisable, but it is obviously C-linearisable using $J = \{0, 1\}$. □

Clearly T1-linearisability is not a necessary condition for T-linearisability. Consider $p \in \mathcal{B}[X, Y]$,

$$p = x_0x_1y_0 + x_0x_1 + x_2 = (y_0 + 1)x_0x_1 + x_2.$$

This polynomial obviously is T-linearisable by setting $y_0 = 1$, but due to the monomial x_0x_1 T1-linearisation does not work here.

Definition 2.4.5. For any index subset $I \subseteq [n]$ and polynomial $p \in B[X, Y]$ we define the set of public monomials of p relative to I as

$$E_p(I) = \{y_J : x_Iy_J \in p\}$$

and the tweaking polynomial $p_{E_p(I)} \in \mathcal{B}[Y]$ as

$$p_{E_p(I)} = \sum_{y_J \in E_p(I)} y_J.$$

Using this we can express any tweakable polynomial $p \in \mathcal{B}[X, Y]$ as

$$p = \sum_{I \subseteq [n]} p_{E_p(I)}x_I \ .$$

In other words, $p_{E_p(I)}$ is the coefficient of x_I in p if seen as $p \in \mathcal{B}[Y][X]$.

Clearly, if we want to obtain a linear polynomial in X, we need to evaluate all $B[Y]$ coefficients of $x_I,\ |I| \geq 2$ to zero. In the following proposition we use this to present an equivalent definition of T-linearisation.

Proposition 2.4.6. *Tweakable polynomial $p \in \mathcal{B}[X, Y]$ is T-linearisable if and only if*

$$\begin{gathered}\exists J \subseteq [m]: \\ (\exists v \in C_J : [\forall I \subseteq [n],\ |I| \geq 2 : p_{E_p(I)|v} = 0 \\ \wedge\ (\exists I' \subseteq [n],\ |I'| = 1 : p_{E(I')|v} \neq 0)])\end{gathered}$$

In other words, a tweakable polynomial is T-linearisable, if there is a solution to the system of polynomial equations yielded by $p_{E_p(I)}$'s, where $|I| \geq 2$ for that some of the $p_{E_p(I')},\ |I'| = 1$ does not evaluate to zero.

Proof. We prove the forward implication by contradiction, the backward directly.
"⇒": Let's assume that p is T-linearisable and

$$\begin{gathered}\forall J \subseteq [m]: \\ \forall a \in \mathbb{F}_2^m\ [(\exists I \subseteq [n], |I| \geq 2 : p_{E_p(I)}(a) = 1) \\ \vee\ (\forall I' \subseteq [n], |I'| = 1 : p_{E(I')}(a) = 0)]\end{gathered}$$

That means that after any partial evaluation in public variables there either remains some monomial that is not linear in secret variables or the resulting polynomial is a constant. Hence the contradiction.
"⇐": If there exists such J and $v \in C_J$, then we can eliminate all the monomials that are non-linear in secret variables by partial evaluation in v and there is at least one monomial, that is linear in secret variables that remains in the polynomial after the partial evaluation. So $p_{|v}$ is a polynomial that is linear in secret variables, hence p is T-linearisable. □

This proposition yields a very compelling way of T-linearising a tweakable polynomial. First, we find the solutions for the system of polynomial equations defined by the $p_{E_p(I)}$'s for $|I| \geq 2$. Then we choose those solutions, for which there exist I' such that $|I'| = 1$ and $p_{E_p(I')}$ is non-zero after the partial evaluation. Naturally, this may be ineffective or even impossible, as shown in following example:

Example 2.4.7. Because our usual example polynomial is, as demonstrated, T-linearisable, for purposes of this example, we present a different polynomial, $q \in \mathcal{B}[X, Y]$, $q = x_0 + x_0x_1y_0 + x_1x_2y_0 + x_1x_2$. Obviously,

$$q = x_0 + x_0x_1 \cdot y_0 + x_1x_2 \cdot (y_0 + 1)$$

which yields an equation system

$$\begin{aligned} y_0 + 1 &= 0 \\ y_0 &= 0 \end{aligned}$$

which has no solution.

Because solving a system of polynomial equations over $\mathbb{F}_2$ in general is computationally ineffective, we present a simpler version, which we might be able to solve in a more efficient manner (given that the system of equations actually has a solution, otherwise we just conclude that there is none).

Proposition 2.4.8. *Let $p \in \mathcal{B}[X, Y]$. If*

$$\forall x_I y_J \in H_p : |J| \leq 1 \wedge$$
$$\wedge \left[\exists a \in \mathbb{F}_2^m : (\forall I' \subseteq [n], |I'| \geq 2 : p_{E(I')}(a) = 0) \wedge (\exists i \in [n] : p_{E(\{i\})}(a) = 1)\right]$$

then p is T-linearisable. Specially, we say that p is T2-linearisable.

This means, that a polynomial is T2-linearisable, if is T-linearisable and the tweaking polynomials for all I's, such that $|I| \geq 2$, are linear or constant.

Proof. Corollary of previous proposition. □

The algorithm Alg. 3 this proposition yields is quite straightforward, but a little more complex, because it consists of three main parts. First, we check if all monomials from H_p are linear or constant in Y. Then we create and solve an equation system consisting of $p_{E(I)} = 0$ equations for all $I \subseteq [n] : \ |I| \geq 2$. Finally we look for a solution of this system that evaluates to one on at least one $p_{E(i)}, \ i \in [n]$.

Algorithm 3 T2-linearisation

```
// Check for linearity
for x_I y_J ∈ H_p do
    if |J| ≥ 2 then return False
    end if
end for
// Create and solve the equation system
EquationSystem = ∅
for I ⊆ [n] : ∃J ⊆ [m] : x_I y_J ∈ H_p do
    EquationSystem.add(p_{E_p(I)} = 0)
end for
Solutions = Solve (EquationSystem)
//Search for a suitable solution
for a ∈ Solutions do
    if ∃i ∈ [n] : p_{E({i})}(a) = 1 then return a
    end if
end for
return False
```

Corollary 2.4.9. *T2-linearisability does not imply T1-linearisability.*

Proof. Polynomial $p = x_0y_0y_1 + x_0x_1y_0 + x_0x_1 = x_0y_0y_1 + x_0x_1(y_0 + 1)$ is clearly T2-linearisable (set $y_0 = y_1 = 1$), but not T1-linearisable. □

Corollary 2.4.10. *T1-linearisability does not imply T2-linearisability*

Proof. Polynomial $p = x_0y_0 + x_0x_1y_0y_1$ is clearly T1-linearisable (set $y_1 = 0$), but not T2-linearisable. □

2.5 TC-linearisation of tweakable polynomials

In this section we present TC-linearisation, our generalisation of Shamir's Cube Attack's C-linearisation.

In order to proceed to the definition of a TC-linearisable polynomial, we first define more general version of a maxterm.

Definition 2.5.1. Let $J \subseteq [m]$ be an index subset. We call the monomial y_J a T1-/T2-/T-maxterm, if the superpoly of J in p is a T1-/T2-/T-linearisable polynomial.

Definition 2.5.2. Let $p \in \mathcal{B}[X, Y]$ be a tweakable polynomial. Then p is TC1-/TC2-/TC-linearisable if and only if

$$\exists J \subseteq [m] : y_J \text{ is a T1-/T2-/T-maxterm respectively.}$$

Note, that TC1-/TC2-/TC-linearisation with $J = \emptyset$ equals T1-/T2-/T-linearisation since $p_{S(\emptyset)} = p$.

Example 2.5.3. In this example we use the same polynomial $p \in \mathcal{B}[X, Y]$ as previously:

$$p = x_0y_0y_1y_2 + x_1y_0y_1y_2 + x_0y_0y_2 + x_0x_1y_0y_1 + x_1x_2y_1 + x_2y_0y_1 + y_0y_1 + y_0y_2 + y_0 + y_2$$

This polynomial is TC-linearisable, because we can either set $y_0 = y_2 = 1$ and obtain a non-constant linear polynomial $x_0 + 1$ by T-linearisation only or get for example $x_0 + x_1$ by summing over the cube defined by $J = \{0, 1, 2\}$. There is also the possibility of using a combination of both, as is made possible using TC-linearisation:

$$p = y_0 \cdot (x_0y_1y_2 + x_1y_1y_2 + x_0y_2 + x_0x_1y_1 + x_2y_1 + y_1 + y_2 + 1) + x_1x_2y_1 + y_2$$

and then we set $y_1 = 0 \wedge y_2 = 1$ to get x_0 as our linear polynomial. In this particular case it would of course be more efficient to use T-linearisation only, because x_0 is obviously linearly dependent on $x_0 + 1$.

Note. We call a tweakable polynomial $p \in \mathcal{B}[X, Y]$ TC1-/TC2-/TC-linearisable, if we can derive a polynomial that is linear in secret variables from it by using partial evaluation as described in the equivalent definitions of T1-/T2-/T-linearisation and cube summation presented as C-linearisation combined.

In this case the algorithmisation is simple again. At first we choose an index subset $J \subseteq [m]$ and then we attempt to linearise the superpoly of J in the polynomial using respective T-linearisation variant. We use either T1- or T2-linearisation, because we have no effective way of solving polynomial systems of equations.

Clearly, if a tweakable polynomial does not contain any monomials linear in secret variables, then we can not apply any of the presented techniques:

Corollary 2.5.4. *Let $p \in \mathcal{B}[X, Y]$ be a tweakable polynomial. If $L_p = \emptyset$, then p is not T-, C- or TC-linearisable.*

2.5.1 Minimizing the cube

Let's have an index subset $J \subseteq [m]$. We now assume, that $p \in \mathcal{B}[X, Y]$, $p(X, Y) = y_J \cdot p_{S(J)}(X, Y) + q_J(X, Y)$ is a TC-linearisable polynomial and that y_J is a T-maxterm. The monomial y_J or, more precisely, the index subset J may not be minimal with this property. We want to have it as small as possible though, because it means summing fewer equations because of smaller cube while summing the partially evaluated polynomials.

The following example illustrates the properties we are looking for when trying to reduce the size of J.

Example 2.5.5. Let $p \in \mathcal{B}[X, Y]$ be the same polynomial as in previous example:

$$p = x_0y_0y_1y_2 + x_1y_0y_1y_2 + x_0y_0y_2 + x_0x_1y_0y_1 + x_1x_2y_1 + x_2y_0y_1 + y_0y_1 + y_0y_2 + y_0 + y_2$$

The obvious first choice of a monomial to factor out of this polynomial to obtain a superpoly is $y_0y_1y_2$:

$$y_0y_1y_2 \cdot (x_0 + x_1) + x_0y_0y_2 + x_0x_1y_0y_1 + x_1x_2y_1 + x_2y_0y_1 + y_0y_1 + y_0y_2 + y_0 + y_2$$

This way we get a superpoly $x_0 + x_1$ by using C-linearisation only. However, we have to sum over a cube of dimension three.

But if we wanted to use TC-linearisation, nothing stand in the way of using y_0y_2:

$$y_0y_2 \cdot (x_0y_1 + x_1y_1 + x_0 + 1) + x_0x_1y_0y_1 + x_1x_2y_1 + y_0y_1 + y_0 + y_2$$

and obtaining non-constant linear polynomial $x_0 + 1$ by setting y_1 zero or $x_1 + 1$ by setting $y_1 = 1$.

We can even use y_0 only:

$$y_0 \cdot (x_0y_1y_2 + x_1y_1y_2 + x_0y_2 + x_0x_1y_1 + x_2y_1 + y_1 + y_2 + 1) + x_1x_2y_1 + y_2$$

and set $y_1 = 0 \wedge y_2 = 1$ to get x_0 as our linear polynomial.

And in this very particular case we do not have to factor out anything, just set $y_1 = 0$:

$$x_0y_0 \cdot 0 \cdot y_2 + x_1y_0 \cdot 0 \cdot y_2 + x_0y_0y_2 + x_0x_1y_0 \cdot 0 + x_1x_2 \cdot 0 + x_2y_0 \cdot 0 + y_0 \cdot 0 + y_0y_2 + y_0 + y_2 = \\ = x_0y_0y_2 + y_0y_2 + y_0 + y_2$$

set $y_0 = y_2 = 1$ and the function we have is $x_0 + 1$ again, using T-linearisation only.

Plainly, we cannot use y_0y_1:

$$y_0y_1 \cdot (x_0y_2 + x_1y_2 + x_0x_1 + x_2 + 1) + x_0y_0y_2 + x_1x_2y_1 + y_0y_2 + y_0 + y_2$$

due to the non-linear monomial x_0x_1 in $p_{S(I)}$ that does not contain any public variable.

Nor can we use y_1 alone:

$$y_1 \cdot (x_0y_0y_2 + x_1y_0y_2 + x_0x_1y_0 + x_1x_2 + x_2y_0 + y_0) + x_0y_0y_2 + y_0y_2 + y_0 + y_2$$

because of x_1x_2 for the same reason as with the previous cube.

But again, y_2 alone:

$$y_2 \cdot (x_0y_0y_1 + x_1y_0y_1 + x_0y_0 + y_0 + 1) + x_0x_1y_0y_1 + x_1x_2y_1 + x_2y_0y_1 + y_0y_1 + y_0$$

and y_1y_2:

$$y_1y_2 \cdot (x_0y_0 + x_1y_0) + x_0y_0y_2 + x_0x_1y_0y_1 + x_1x_2y_1 + x_2y_0y_1 + y_0y_1 + y_0y_2 + y_0 + y_2$$

are perfectly fine too, because we now have $x_1 + 1$ by setting $y_0 = y_1 = 1$ and $x_0 + x_1$ by $y_0 = 1$ respectively.

However improbable this situation can be, we get many ways of choosing the cube variables. In this case, when efficiency is key, we would naturally go for the empty cube.

When outlining the conditions of the minimality of index subsets for TC-linearisation, we begin with the C-linearisation to illustrate the matter.

Definition 2.5.6. For any index subset $J \subseteq [m]$ such that y_J is a maxterm we define the C-minimal-candidate subset $J' \subseteq J$ as a minimal one in terms of the count of its elements while $y_{J'}$ remains a maxterm.

For respective versions of TC-linearisation we define the minimal subsets analogically. But before we do that, we present a lemma as to what makes an index subset a C-minimal-candidate:

Lemma 2.5.7. *Let y_J be a maxterm. The index subset J is a C-minimal-candidate if and only if*

$$\forall J' \subsetneq J \; \exists h \in H_p : y_{J'} | h$$

Proof. Trivial corollary of the proposition outlining the C-linearisation conditions. □

Based on this, we can define a C-minimal subset:

Definition 2.5.8. We define the C-minimal subset as a minimal one in terms of the count of its elements among the C-minimal-candidates.

The rules for T1-linearisability are clear and quite simple, so we can actually define a TC1-minimal subset as well:

Definition 2.5.9. For any index subset $J \subseteq [m]$ such that y_J is a T1-maxterm we define the TC1-minimal candidate subset $J' \subseteq J$ as a minimal one in terms of the count of its elements while $y_{J'}$ remains a T1-maxterm.

The following proposition provides constructive guide to determining, whether an index subset is a TC1-minimal candidate.

Proposition 2.5.10. *Let y_J be a T1-maxterm. The index subset J is a TC1-minimal-candidate if and only if*

$$\forall (J' \subsetneq J : \exists s \in H_p : y_{J'} | s \wedge y_J \not| s)$$

Basically, an index subset stops defining a T1-maxterm precisely at the moment when such a monomial "slips" into the superpoly, such that it cannot be eliminated by partial evaluation as described in the equivalent definitions of T1-linearisation.

Proof. The implication "$\Leftarrow$" is trivial, we prove the other by contradiction. "$\Rightarrow$": If for some $J' \subsetneq J$ existed such y for every $s \in H_p$, $y_j | s \wedge y_J \not| s$, we could use J', that is smaller than J which contradicts J's minimality. □

Definition 2.5.11. We define a TC1-minimal subset as a minimal one in the count of its elements among the TC1-minimal candidates.

Example 2.5.12. We shall now examine the different superpolys we got in the previous example. To be as efficient as possible, we use the smallest cubes we can so that the superpolys we get are linearly independent:

- $y_I = 1$: $p_{|y_1=0, y_0=y_2=1} = x_0 + 1$
- $y_I = y_2$: $p_{S(I)} = x_1 + 1$

These are obviously linearly independent and given an RHS bit we can solve the equation system, leaving us with the need to guess x_2. In this case we could even solve the equations independently.

Now we present the same definition and proposition for T2 linearisation and we shall see, whether we can get any further using it. Since we are unable to tell, whether a polynomial really is T2-linearisable without actually solving the system of equations, we can only determine, whether the system of equations fulfills the basic conditions of the T2-linearisation. Thus we present yet the following definitions:

Definition 2.5.13. We call a monomial $y_J, J \subseteq [m]$ a T2-maxterm candidate, if the system of polynomial equations yielded by the respective tweaking polynomials is linear.

Definition 2.5.14. For any index subset $J \subseteq [m]$ such that y_J is a T2-maxterm candidate we define the TC2-attempt-minimal-candidate subset $J' \subseteq J$ as a minimal one in terms of the count of its elements while $y_{J'}$ remains a T2-maxterm candidate.

Proposition 2.5.15. *Let y_J be a T2-maxterm candidate. The index subset J is TC2-attempt-minimal-candidate if and only if*

$$\forall J' \subsetneq J \; \exists y_{J''} x_I \in H_{p_{S(J')}} : |J''| \geq 2.$$

In other words, an index subset stops defining a T2-maxterm candidate when there is such a polynomial in the set of tweaking polynomials, that it does not fulfill the T2-linearisation conditions.

Proof. We prove the forward implication by contradiction, the other directly.
"⇒": We assume, that J is T2-attempt-minimal and

$$\exists J' \subsetneq J : (\forall y_{J''} x_I \in H_{p_{S(J')}} : |J''| \leq 1)$$

But that implies, that there is another, smaller subset of J, namely J', which means, that J was not minimal after all.
"⇐": If the RHS holds, for any $J' \subsetneq J$ we get a system of equations that does not fulfill the T2-linearisation conditions, which means, that J really is TC2-attempt-minimal. □

At last, we define the TC2-attempt minimal subset:

Definition 2.5.16. We call $J \subseteq [m]$ TC2-attempt-minimal, if J is a smallest TC2-attempt-minimal-candidate in terms of the count of its elements.

Chapter 3

Linearising Trivium keystream polynomials

With the previous one dealing with the underlying theory, this chapter describes the experimental part of the thesis. At first we present in detail the description of the cryptosystem, a derived variant of which we take on. It is Trivium, a cipher presented within the eSTREAM project, that got into the final phase. The original description is to be found in [2], we have modified it to fit our purpose and notation.

After that we present the attack itself. Since we in this case only wanted to test our linearisation methods, we have not attempted the key-recovery, which is the aim of the full attack. In other words we show, how far we got with the techniques devised by us and compare them to the Cube Attack.

3.1 Trivium

Before we present the attack itself, we describe the cryptosystem we are about to attack. It is a reduced variation of a stream cipher called Trivium [2]. At first we describe the original cryptosystem and after that we describe reduced variation we will attack.

3.1.1 Trivium cipher description

Trivium is a very simple stream cipher with three non-linear feedback registers, 80 bit key and 80 bit initialisation vector (IV). The cipher produces a keystream $\{z_i\},\ z_i \in \mathbb{F}_2$ which is added to the plaintext to produce the ciphertext.

The three registers of Trivium are of lengths 93, 84 and 111 respectively and form the inner state of the cipher. The state is updated every step and we will use the notation $S_i,\ \in \mathbb{F}_2^{288}$ to denote the state at the beginning of the i-th step for $i \geq 0$:

$$S_i = (s_{i,0}, ..., s_{i,92}, \\ s_{i,93}, ..., s_{i,176}, \\ s_{i,177}, ..., s_{i,287}).$$

$(s_{i,0}, ..., s_{i,92})$ is the first register, $(s_{i,93}, ..., s_{i,176})$ the second and $(s_{i,177}, ..., s_{i,287})$ the third. The cipher defines a state update function $F : S_i \mapsto S_{i+1}$ which is a non-linear function that shifts all three registers by one to the left and sets the first bits of the registers to $f_3(S_i),\ f_1(S_i), f_2(S_i)$. I.e.

$$S_{i+1} = (f_3(S_i), s_{i,0}, ..., s_{i,91}, \\ f_1(S_i), s_{i,93}, ..., s_{i,175}, \\ f_2(S_i), s_{i,177}, ..., s_{i,286})$$

where

$$f_1(S_i) = s_{i,65} + s_{i,92} + s_{i,90} \cdot s_{i,91} + s_{i,170}$$
$$f_2(S_i) = s_{i,161} + s_{i,176} + s_{i,174} \cdot s_{i,175} + s_{i,263}$$
$$f_3(S_i) = s_{i,242} + s_{i,287} + s_{i,285} \cdot s_{i,286} + s_{i,68}$$

are the feedback function.

The cipher's operation can be divided into two parts: the initialization algorithm and the keystream generation algorithm. The initialisation starts by setting the initial inner state $S_0 = (s_{0,0}, \ldots, s_{0,287})$ as

$$s_{0,0} = k_0, \ldots, s_{0,79} = k_{79}, \quad s_{0,93} = IV_0, \ldots, s_{0,172} = IV_{79}, \quad s_{0,285} = s_{0,286} = s_{0,287} = 1$$

and the all the other $s_{0,i} = 0$. I.e. the key $(k_0, \ldots, k_{79})$ is loaded into the beginning of the first register, the $IV = (IV_0, \ldots, IV_{79})$ into the beginning of the second register and the last three bits of the third register are set to one. Then it updates the state 1152 times to produce the state S_{1152} in the end. Afterwards, the keystream generation starts. On each step $i \geq 1152$, the keystream bit is generated as

$$z_i = s_{i,65} + s_{i,92} + s_{i,161} + s_{i,1768} + s_{i,242} + s_{i,287} \tag{3.1.1}$$

and then the state S_i is updated by the state-update function F to S_{i+1}. The cipher is depicted in Figure 3.1.

Let us denote the key variables by $X = (x_0, \ldots, x_{79})$ and IV by $Y = (y_0, \ldots, y_{79})$. Then, for each bit $s_{i,j}$ of the inner state S_i there exists a polynomial $g_{i,j} \in \mathcal{B}[X, Y]$ such that

$$s_{i,j} = g_{i,j}(K, IV).$$

Similarly, for each keystream bit z_i there exist a polynomial $p_i \in \mathcal{B}[X, Y]$ such that

$$z_i = p_i(K, IV)$$

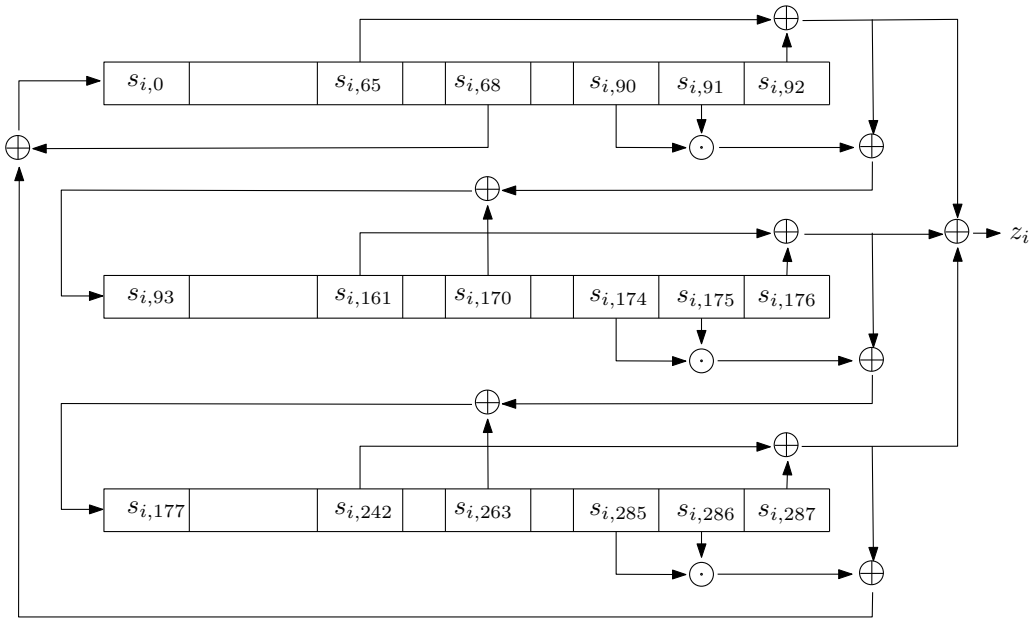


Figure 3.1: Stream cipher Trivium

and by using (3.1.1) we get

$$p_i(X,Y) = g_{i,65}(X,Y) + g_{i,92}(X,Y) + g_{i,170}(X,Y) + g_{i,176}(X,Y) + g_{i,242}(X,Y) + g_{i,287}(X,Y).$$

So for an unknown key X and known initialisation vector IV, an attacker can create a system of polynomial equations $z_i = p_i(X, IV)$ in X by observing the keystream z_i. In Trivium, the initialisation has 1152 steps, hence the attacker knows z_i, $i \geq 1152$. Intuitively, the shorter the initialisation, the easier it should be to attack the cipher as polynomials $p_i(X,Y)$ become more complex with increasing i.

In order to demonstrate the capabilities of the presented techniques we will assume that the keystream generation starts already from the state S_0, i.e. the keystream is $\{z_i\}_{i\geq 0}$.

3.1.2 Trivium-8

Trivium-8 is a version of Trivium with shortened key and IV to 8 bits each. Aside from the shortened key and IV, it is the same Trivium as described above, so it has a 288 bit inner state. The length of the key and IV was chosen to be 8, i.e. $K = (k_0, \ldots, k_7)$ and $IV = (IV_0, \ldots, IV_7)$, since this should be enough to make the polynomials $p_i(X,Y)$ reasonably complex (interesting) while maintaining them small enough for the computation to be feasible with a standard PC.

3.2 Attack description

As already mentioned, we will use the described linearisation methods to attack Trivium-8, a Trivium reduced in key and IV bits.

We assume that an attacker has access to Trivium-8 with fixed unknown key K. He can repeatedly choose the IV and obtain the keystream $z_i = p_i(K, IV)$ (chosen IV attack). His goal is to find K by solving the respective equation systems.

For this purpose we need to compute the polynomials $g_{i,j}(X, Y)$ for all inner state bits and use them to compute the polynomials $p_i(X, Y)$ expressing the keystream bits. To obtain the actual values of z_i, we need an implementation of Trivium-8 that computes the keystream bits from the K and IV.

3.2.1 Attack using T-linearisation

This part is pretty straightforward. Assume that $IV \in \mathbb{F}_2^m$ and $i \geq 0$ such that $p_i(X, IV)$ is linear in X. Then we get the value of $z_i = p_i(K, IV)$ and form a linear equation $p_i(X, IV) = z_i$.

If the J from the definition of T-linearisation is such that $J \neq [m]$, we add to the values of respective $v \in C_J$ values for the remaining public variables arbitrarily.

3.2.2 Attack using C-linearisation

Assume that for an $J \subseteq [m]$ the y_J is a maxterm of $p_i(X, Y)$ so by assigning ones to all $y_j,\ j \notin J$ we obtain a linear superpoly. We set $U = \{(u_0, \ldots, u_{m-1}) \in \mathbb{F}_2^m; u_j = 1\ \forall j \notin J\}$ and obtain $z_i(a) = p_i(K, u)$ for all $u \in U$. The equation obtained by C-linearisation is then

$$p_{S(J)}(X, v) = \sum_{u \in U} z_i(u)$$

where v is an element of U. Note that $p_{S(J)}(X, Y)$ can depend only on public variables with indices not in J, hence the equation does not depend on the particular choice of $v \in U$.

3.2.3 Attack using TC-linearisation

In C-linearisation, we set all public variables with indices not in J to one. In TC-linearisation, we assume that there exists $w \in C_{[m]\setminus J}$, such that $p_{S(J)|w}$ is linear. I.e. when summing over all k-tuples from the cube C_J, we have to extend each k-tuple with the T-linearising bits for partial evaluation, that remain fixed during the whole summing.

3.3 Experimental results

In this chapter we present results we got when we applied the presented linearisation techniques on reduced Trivium as presented in respective chapter.

Since solving a system of linear equations is simple, in our experiments we shall concentrate on whether we can actually obtain any linear, non-constant polynomials, from which we could build such.

For polynomial multiplication to build the representation we used a variant of algorithm from [4].

3.3.1 C-linearisation

The C-linearisation proved to be, as expected, very effective. We could use it to linearise the polynomials representing keystream bits with indices up to 609. It is important to note, that by no means all keystream polynomials up to the 610^{th} are C-linearisable. Moreover, although the aforementioned polynomial technically is C-linearisable, it is highly ineffective, because the cube contains all 8 public variables. That results in the complexity of summing over the cube being approximately the same as of exhaustive search for the key. The last polynomial, for which C-linearisation is actually useful, is that with index 582. For the polynomials representing up to 575^{th} keystream bit there is a lot of maxterms to be found. However, after the half of the full initialisation (576 rounds) the C-linearisable polynomials are starting to appear rather sparsely and after 609 rounds they stop appearing altogether.

3.3.2 T1-linearisation

T1-linearisation did not prove to be especially effective. We could T1-linearise polynomials representing up to the 361^{st} keystream bit.

This is actually a result we expected: If T1-linearisation would be effective on the polynomials, even if in relatively early stages of the initialisation, it would mean, that there would be no monomial $x_I y_\emptyset$, $I \subseteq [n]$. This is highly improbable though, because it would mean that none of a set of 2^n monomials would be present, which happens with probability of 2^{-2^n} if dealing with a random polynomial, which we assume the polynomials in the later stages of initialisation to be.

3.3.3 T2-linearisation

In spite of T2-linearisation using an approach that significantly differs from that used in T1-linearisation, we managed to linearise using it keystream polynomials up to p_{361}. It is easy to see, why this technique was not any more successful: it demands, that for the polynomial $p \in \mathcal{B}[X, Y]$ that we are attempting to linearise every monomial in H_p has degree at most one in secret variables. In fact then, this is a surprisingly good result.

3.3.4 TC1-linearisation

With TC1-linearisation, the situation is a bit more complicated. It is at least as effective as C-linearisation, but it could at some point prove to be able to reduce the cube and

therefore the complexity of the linearisation. This happened only when the polynomial was T1-linearisable, so this technique turned out to be a bit disappointing.

3.3.5 TC2-linearisation

As we already hinted, TC2-linearisation is the technique, that really does improve the C-linearisation. We managed to linearise polynomials representing the keystream of Trivium-8 with indices up to 622, which is slightly more, than we managed using C-linearisation (609). Despite the TC2-linearisable polynomials not appearing significantly more often than the C-linearisable ones, the cubes to sum over in TC2-linearisation are, especially for the later keystream polynomials, smaller than those for C-linearisation. This proves our hypothesis, that TC2-linearisation is more efficient than C-linearisation. Good example of this is the $p_{622}(X, Y)$, which is C-linearisable with the cube of dimension 8, i.e. the maximal one, but there are T2-maxterms with degree 6, i.e. the dimension of the cube is 6.

The linearisation attempts we made have rather met our expectations. Neither T1-, nor T2-linearisation proved to be effective in any interesting extent. Also TC1-linearisation did not bring any improvement over C-linearisation beyond the union with T1-linearisation. However, aside from the expected success of C-linearisation, which surprised by being able to linearise as far as the keystream polynomial with index 609, we were able to linearise the polynomial representing the 623^{rd} keystream bit ($p_{622}(X, Y)$) using TC2-linearisation, which we consider to be a success, since it is a clear improvement over C-linearisation ($p_{609}(X, Y)$). Moreover, the C-linearisability of the polynomial representing the 610^{th} keystream bit has an ambivalent meaning in terms of an attack, the last C-linearisable polynomial, that could actually be useful for an attack is that with index 582. In figure 3.2 we show the results graphically.

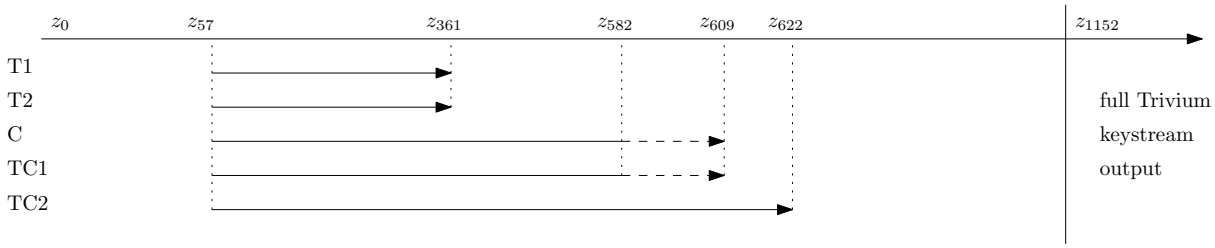


Figure 3.2: Range of effectiveness of linearisation techniques

Chapter 4

Conclusion

In this thesis we presented a detailed description of Cube Attack devised by Adi Shamir et. al. and its generalisation that proved to be slightly more effective when tested on key- and IV-reduced Trivium variant. None of these techniques is advanced enough to linearise keystream polynomials after full 1152 initialisation rounds, but the fact, that well after the initialisation there remain some monomials that are linear in secret variables hint, that under some circumstances and using more advanced techniques, that are to be discovered, could be linearised, or at least significantly simplified in similar manner too.

These more advanced techniques are subject of further research of ours. We already have some ideas, where we could be looking for improvements, that might stretch the effectiveness well beyond the current 622 initialisation rounds of reduced Trivium. Naturally, its application on full Trivium or another cryptosystem is an obvious aim of all these efforts. One way to attempt that lies in a more detailed analysis of the polynomials for key- and IV-reduced Trivium, which could yield some interesting properties of the original cryptosystem.

Bibliography

[1] Claude Carlet. Boolean functions for cryptography and error correcting codes. Chapter of the monography "Boolean Models and Methods in Mathematics, Computer Science, and Engineering", Cambridge University Press, 2006. http://www.math.univ-paris13.fr/~carlet/chap-fcts-Bool-corr.pdf.

[2] Christophe de Cannière and Bart Preneel. Trivium. eSTREAM: the ECRYPT Stream Cipher Project, 2005. http://www.ecrypt.eu.org/stream/.

[3] Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. Cryptology ePrint Archive, Report 2008/385, 2008. http://eprint.iacr.org/.

[4] Subhabrata Samajder and Palash Sarkar. Fast multiplication of the algebraic normal forms of two boolean functions. International Workshop on Coding and Cryptography 2013, Bergen (Norway), 2013. http://www.selmer.uib.no/WCC2013/.

List of Figures