

# Some positive and negative results on Cube Attacks

Thomas Dullien, Ruhr-University Bochum

January 14, 2009

# Talk overview

This talk will consist of three parts:

- ▶ Introduction to Cube attacks
- ▶ An improvement to cube attacks in the non-black-box scenario
- ▶ A result that shows that obvious generalizations of cube yield little benefit

# Introduction to Cube

Idea: The cipher as an oracle

- ▶ Cube: A generic method for solving certain nonlinear polynomial systems over  $\mathbb{F}_2$
- ▶ Important idea: Model cipher differently
  - ▶ Equations that describe the cipher mix plaintext and key bits
  - ▶ Given a chosen-plaintext-scenario or chosen-IV-scenario, the attacker can *evaluate* the equations with unknown key bits and chosen plaintext bits
  - ▶ This is very different from the usual “solve equations” approach

# Tweakable Variables

The attacker-controlled parts are called *tweakable*

The cipher is given by

$$f_1, \dots, f_n \in \mathbb{F}_2[t_1, \dots, t_n, k_1, \dots, k_n]$$

The attacker has oracle access to the cipher: He may choose  $t \in \mathbb{F}_2^n$  for the variables  $t_1, \dots, t_n$  and have the cipher give him  $c_{t,i} \in \mathbb{F}_2$  with

$$f_i\left(\underbrace{t_1, \dots, t_n}_{\text{attacker-controlled}}, \underbrace{k_1, \dots, k_n}_{\text{unknown to the attacker}}\right) = c_{t,i}$$

## Definition

*Tweakable:* The attacker-controlled bits  $t_1, \dots, t_n$  are called *tweakable variables*. A function with such variables is called *tweakable*.

# Factoring out a linear term

...that is then used for solving...

Given a monomial  $m$  in tweakable variables and a function  $f$  we can write:

$$f = m/l + r$$

where

$$l = \sum_{l' \in f, m | l'} \frac{l'}{m}, \quad r = \sum_{r' \in f, m \nmid r'} r'$$

If  $\deg l = 1$ , we call  $m$  a *maxterm*.

Let  $m \in \mathbb{F}_2^{\deg m}$ . In the following, I write  $f(m)$  to mean:  $f$  evaluated with the  $i$ -th variable in  $m$  assigned to the  $i$ -th entry of  $m$ .

# Isolating the linear term

The oracle can help us extract  $l$

## Theorem

*Let  $f$  be factored into  $f = m l + r$ . Let  $M$  be the set of all  $2^{\deg m}$  possible assignments to the variables in  $m$ . Then it holds that*

$$\sum_{m \in M} f(m) = \sum_{m \in M} m(m)l + r(m) = l$$

*Since  $m$  evaluates to 1 on exactly one value in  $M$ , it is clear that  $l$  will be part of the result. Why do the  $r(m)$  cancel out?*

# Isolating the linear term

Exposing  $l$

Proof.

$r$  can be divided into three types of monomials  $r_i$ :

1.  $r_i | m$ , e.g. all variables in  $r_i$  are in  $m$ . This means that  $r_i$  evaluates to 1 an even number of times in our sum, hence cancels.
2.  $r_i$  does not have any variables in common with  $m$ . This means that  $r_i$  occurs an even number of times in our sum, hence cancels.
3.  $r_i$  shares some, not all, variables with  $m$ . Since the shared part will evaluate to 1 an even number of times, the non-shared part will occur an even number of times in the sum, and thus cancel.

Hence  $r$  cancels in the sum. Since  $m$  evaluates to 1 only on the  $(1, \dots, 1)$ -vector,  $l$  remains. □

# Some more facts

...from the original paper...

The idea of the attack is then to

1. Extract linear terms from the equations
2. Solve the linear equation system

Some other facts about cube attacks:

- ▶ *Most important:* They can be performed without  $f$  being explicitly given (e.g. black-box)
- ▶ It costs  $2^{\deg m}$  oracle queries to isolate  $l$
- ▶ They require pretty tight degree bounds to work reliably

# Three results (one positive, two negative) on Cube

Fewer queries, generic versions do not help much

This short talk presents two results for Cube-style attacks.

- ▶ Positive: For an explicitly given decomposition  $f = ml + r$ , one needs to sum over way fewer than  $2^{\deg m}$  values (only in the white box scenario !)
- ▶ Negative: Even though we can always find polynomials  $p, r$  so that  $f = pl + r$  with  $\deg l = 1$ , they require extra properties for cube-style summations

# Optimizing queries for Cube-style attacks

We need fewer queries than  $2^m$

The following will make the (unlikely) assumption that one has a *non-black-box*, tweakable polynomial explicitly given. This polynomial is named  $f$ .

- ▶ Cube-attacks decompose  $f = ml + r$  and sum over all possible values for  $m$
- ▶ Is that really necessary ? If  $\deg m$  is large, this is expensive.
- ▶ *No !* (if  $f$  is given explicitly)

One can greatly diminish the number of items to sum over.

# Query Sets

## Definition

Given  $f$  and a decomposition  $f = ml + r$ , a query set  $Q_{f,l}$  is a subset of  $\mathbb{F}_2^{\deg m}$  so that

$$\sum_{m \in Q_{f,l}} f(m) = l$$

# What do we want from a query set ?

What properties does our “query set”  $Q$  need so we can isolate  $l$  ?

We want  $Q$  to satisfy

$$\sum_{x \in Q} f(x) = l$$

- ▶  $m$  needs to evaluate to one an uneven number of times
- ▶  $r$  needs to cancel - e.g. each monomial in  $r$  needs to occur an even number of times

How do we construct such a set ?

## We need the all-one vector, and $r$ must disappear

Clearly, we need the all-one-vector to make  $m$  evaluate to one.  
Hence

$$\begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \in Q$$

Which other elements do we need ?

- ▶ For monomials in  $r$  without tweakable variables: They need to occur an even number of times, so we only need  $|Q|$  to be even
- ▶ For monomials which contain tweakable variables: We need to make sure that the tweakable component evaluates to 1 an even number of times in our set. *This is a property only on the tweakable parts of each monomial – the non-tweakable parts do not matter !*

What's the next step ?

# POsets from polynomials

Consider the set of all tweakable monomials in  $r$ . Add to this set all the “tweakable parts” of the mixed monomials in  $r$ . The result is a set of monomials in tweakable variables.

- ▶ Divisibility of monomials provides us with a partial order
- ▶ We can draw a “hasse diagram” from this

An example would be helpful.

## Only the tweakable parts are relevant

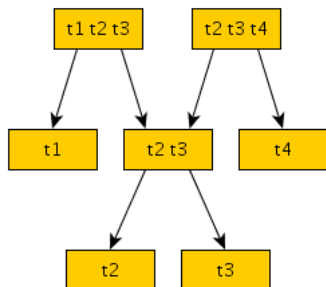
Let's take an ugly boolean polynomial in tweakable  $t_i$  and fixed  $k_i$ .  
For readability, multi-indices are used:

$$\begin{aligned} & t_{1,2,3,4} k_1 k_3 k_8 k_7 + t_1 k_9 k_1 + t_2 k_3 + \dots \\ = & t_{1,2,3,4} ( \\ & (t_1(\dots) + t_2(\dots) + \\ & t_3(\dots) + t_4(\dots) + \\ & t_{1,2,3}(\dots) + t_{2,3,4}(\dots)) \end{aligned}$$

What matters now are thus  $t_1, t_2, t_3, t_4, t_{1,2,3}, t_{2,3,4}$ .

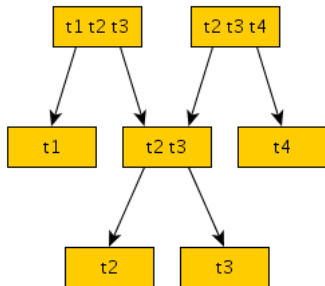
# The resulting Hasse Diagram

If one considers the divisibility relation, the resulting poset has the following Hasse diagram:



If  $m_1|m_2$  and  $m_2$  is one, then  $m_1$  is one

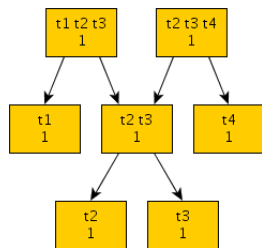
- ▶ Consider  $m_1, m_2$  with  $m_1|m_2$ . Then if  $m_1(x) = 0 \Rightarrow m_2(x) = 0$  and  $m_2(x) = 1 \Rightarrow m_1(x) = 1$
- ▶ We thus have an implication that “travels down” our Poset of monomials



Label all fields with the number of times they evaluate to 1 on  $Q$ .

# The first step of the algorithm

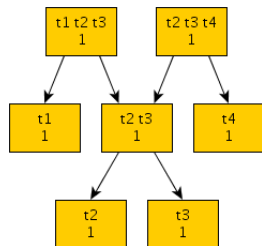
The result of this is:



Our “query set” is currently

$$Q = \left\{ \left( \begin{array}{c} 1 \\ \dots \\ 1 \end{array} \right) \right\}$$

## Adding new queries

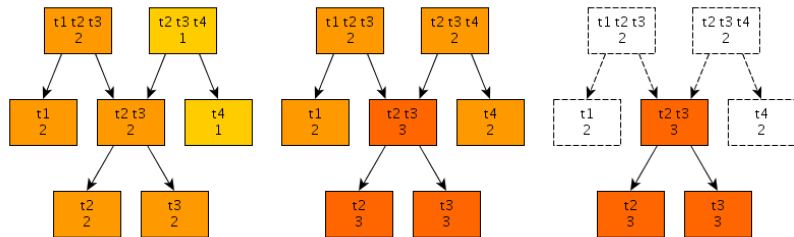


For each monomial that is a “maximal” element in our poset, add a vector with 1-entries for each variable in that monomial to  $Q$ .

$$Q = \left\{ \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} \right\}$$

## Remove the even parts and repeat

The implication  $m_2(x) = 1 \Rightarrow m_1(x) = 1$  travels down the poset, so we can update the labels accordingly:



After the poset is updated, recursively remove all elements with even label and indegree zero.

Wash, rinse, repeat.

# Optimizing queries for Cube-style attacks

We need fewer queries than  $2^m$

Result: Instead of  $2^4 = 16$  queries, we just need 4:

$$Q = \left\{ \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \right\}$$

## So how many queries do we need ?

- ▶ The number of queries is trivially upper bounded by the number of tweakable components in  $r$ , usually  $2^{m-1}$  (yes, I know this is boring)
- ▶ For truly random polynomials, one can show that one should expect less than  $2^{m-2}$
- ▶ Unfortunately, empirically it appears to be  $[2^{m-3}, 2^{m-2}]$
- ▶ This is not fantastic, but results on fully random polynomials are always difficult (MQ is NP-hard)
- ▶ It is easy to construct polynomials that need tiny amounts of queries

## What about structured polynomials ?

- ▶ Take polynomials after 2 rounds of PRESENT (lightweight block cipher)
- ▶ Calculate query set assuming the plaintext bits are tweakable

Bit Number	# of tweakable bits	Queryset size
0-4	16	25+1
4-7	16	109+1
8-11	16	165+1
12-15	16	109+1
16-19	16	279+1
20-23	16	3828+1+0
24-27	16	2381+1
28-31	16	3828+1+0
32-35	16	203+1
36-39	16	2645+1
...	...	...

# What if there are no degree bounds ?

If we cannot have the degree boundaries on  $d$ —random polynomials that are introduced, the probability of a maxterm existing are slim to nonexistent.

## Theorem

*The probability that a fully random tweakable polynomial with  $t$  tweakable and  $c$  fixed variables contains a maxterm of degree  $d$  is approximately:*

$$\frac{2^{2^{c+t} - \sum_{i=d}^t \binom{t}{i}} 2^c}{2^{2^{c+t}}}$$

*This means for degree  $t$  it is  $2^{-c}$ , and gets much worse as the degree decreases.*

# Generalized Cube-style attacks

Do we need to use a monomial ?

This leads us to the following:

- ▶ Decompositions into  $f = ml + r$  and  $\deg l = 1$  don't usually exist for high-degree  $f$
- ▶ How can we work with high-degree  $f$  ?
- ▶ Will a generalization to  $f = pl + r$  with  $p$  a polynomial help us ?
- ▶ Simple fact: For *any* given  $f, l$  one can calculate  $p, r$  with  $f = pl + r$ . Does this mean we can always get a linear term, irrespective of the degree of  $f$  ?

# Generalized Cube-style attacks

Do we need to use monomial ?

Let  $f$  be given. Choose an arbitrary linear term  $l$  and an arbitrary polynomial  $p$ .

It is true (and a bit tautological) that

$$f = \underbrace{f + pl + pl}_{:=r} \Rightarrow f = pl + r$$

We now have a decomposition in order to eliminate  $l$ .  
(I know that this looks like a bad magic trick)

# Generalized Cube-style attacks

Do we need to use monomial ?

- ▶ We hence would want  $f + pl$  to disappear, but  $pl$  not to disappear
- ▶ This means all tweakable components that are present in  $f + pl$  and  $pl$  are forced to be eliminated
- ▶ This means we can only eliminate  $l$  if the tweakable components of  $pl$  are a true subset of the tweakable monomials in  $f$
- ▶ Bad luck: We need a decomposition into disjoint sets of monomials, the “generic” decomposition will not work

⇒ being able to eliminate a linear factor is the exception, not the rule

# Generalized Cube-style attacks

## Summary

- ▶ If we have a non-black-box situation and a maxterm  $m$  of degree  $\deg m$  isolating the linear part is at least a quarter cheaper than  $2^{\deg m}$
- ▶ “Generic” decompositions  $f = ml + r$  are possible, but the Cube algorithm imposes side conditions on them, which make them useless
- ▶ Strong belief (proof is not complete yet): Polynomial versions

$$f = pl + r$$

yield *no benefit* over the monomial version (It appears that existence of such a decomposition implies that each monomial in  $p$  is a maxterm)

# Generalized Cube-style attacks

## Outlook

What else can we still do with Cube ?

- ▶ Low-degree systems should break easily
- ▶ High-degree systems will almost certainly not allow isolating a linear component
- ▶ Generalisation to polynomials won't help
- ▶ (Over-)approximation of  $f$  be useful: Solve the approximation, not the equation. Algebraic Immunity comes into play.
- ▶ Core problem: Without degree boundaries, the isolated polynomials will quickly look random  $\Rightarrow$  we can't solve them

# Generalized Cube-style attacks

Questions ? Code ?

- ▶ SAGE/Python implementation of the queryset-calculation code is available by mailing [thomas.dullien@ruhr-uni-bochum.de](mailto:thomas.dullien@ruhr-uni-bochum.de)
- ▶ Questions ?