ElimLin Algorithm Revisited

Nicolas T. Courtois[1], Pouyan Sepehrdad[2] *, Petr Sušil[2] **, and Serge Vaudenay[2]

[1] University College London, UK
[2] EPFL, Lausanne, Switzerland
n.courtois@ucl.ac.uk,
{pouyan.sepehrdad, petr.susil, serge.vaudenay}@epfl.ch

Abstract. ElimLin is a simple algorithm for solving polynomial systems of multivariate equations over small finite fields. It was initially proposed as a single tool by Courtois to attack DES. It can reveal some hidden linear equations existing in the ideal generated by the system. We report a number of key theorems on ElimLin. Our main result is to characterize ElimLin in terms of a sequence of intersections of vector spaces. It implies that the linear space generated by ElimLin is invariant with respect to any variable ordering during elimination and substitution. This can be seen as surprising given the fact that it eliminates variables. On the contrary, monomial ordering is a crucial factor in Gröbner basis algorithms such as F4. Moreover, we prove that the result of ElimLin is invariant with respect to any affine bijective variable change. Analyzing an overdefined dense system of equations, we argue that to obtain more linear equations in the succeeding iteration in ElimLin some restrictions should be satisfied. Finally, we compare the security of LBlock and MIBS block ciphers with respect to algebraic attacks and propose several attacks on Courtois Toy Cipher version 2 (CTC2) with distinct parameters using ElimLin.

Keywords: block ciphers, algebraic cryptanalysis, systems of sparse polynomial equations of low degree

[Breaking a good cipher should require]
"as much work as solving a system of simultaneous equations
in a large number of unknowns of a complex type."

Claude Elwood Shannon [46]

1 Introduction

Various techniques exist in cryptanalysis of symmetric ciphers. Some involve statistical analysis and some are purely deterministic. One of the latter methods is *algebraic attack* recognized as early as 1949 by Shannon [46]. Any algebraic attack consists of two distinct stages:

- Writing the cipher as a system of polynomial equations of low degree often over $\mathsf{GF}(2)$ or $\mathsf{GF}(2^k)$, which is feasible for any cipher [49, 21, 43].
- Recovering the secret key by solving such a large system of polynomial equations.

Algebraic attacks have been successful in breaking several stream ciphers (see [1, 19, 12, 25, 20, 15, 24, 11] for instance) and a few block ciphers such as Keeloq [38] and GOST [16], but they are not often as successful as statistical attacks. On the other hand, they often require low data complexity. This is not the case for statistical attacks.

General purpose algebraic attack techniques were developed in the last few years by Courtois, Bard, Meier, Faugère, Raddum, Semaev, Vielhaber, Dinur and Shamir to solve these systems [17,

* This work has been supported in part by the European Commission through the ICT program under contract ICT-2007-216646 ECRYPT II.
** Supported by a grant of the Swiss National Science Foundation, 200021_134860/1.

22, 21, 19, 12, 31, 32, 45, 48, 24, 25]. The problem of solving such polynomial systems of multivariate equations is called MQ problem and is known to be NP hard for a random system. Currently, for a random system in which the number of equations is equal to the number of unknowns, there exists no technique faster than an exhaustive key search which can solve such systems. On the other hand, the equations derived from symmetric ciphers turn out to be overdefined and sparse for most ciphers. So, they might be easier to solve. This sparsity is coming from the fact that due to the limitations in hardware and the need for lightweight algorithms, simple operations arise in the definition of cryptosystems. They are also overdefined due to the non-linear operations.

The traditional method for solving overdefined polynomial systems of equations are known to be various Gröbner basis algorithms such as Buchberger algorithm [10], F4 and F5 [31, 32] and XL [22]. The most critical drawback of the Gröbner basis approach is the elimination step where the degree of the system increases. This leads to an explosion in memory space and even the most current efficient implementations of Faugère algorithm [31, 32] under PolyBoRi framework [8] or Magma [41] are not capable of handling *large* systems of equations efficiently. On the other hand, they are faster than other methods for overdefined dense systems or when the equations are over $\mathsf{GF}(q)$ where $q > 2$. In fact, together with SAT solvers, they are currently the most successful methods for solving polynomial systems.

Nevertheless, due to the technical reasons mentioned above, the system of equations extracted from symmetric ciphers turns out to be sparse. Unfortunately, the Gröbner basis algorithms can not exploit this property. In such cases, algorithms such as XSL [21], SAT solving techniques [4, 28, 3], Raddum-Semaev algorithm [45] and ElimLin [17] are of interest.

In this paper, we study the elimination algorithm ElimLin that falls within the remit of Gröbner basis algorithms, though it is conceptually much simpler and is based on a mix of simple linear algebra and substitution. It maintains the degree of the equations and it does not require any fixed ordering on the set of all monomials. On the contrary, we need to work with ad-hoc monomial orderings to preserve the sparsity and make it run faster. This simple algorithm reveals some hidden linear equations existing in the ideal generated by the system. We show in Sec. 7 that ElimLin does not find all such linear equations.

As far as the authors are aware, no clue has been found yet which demonstrates that ElimLin at some stage stops working. This does not mean that ElimLin can break any system. As mentioned earlier, for a random system this problem is NP hard and Gröbner basis algorithms behave much better for such dense random systems. But, the equations derived from cryptosystems are often not random (see [33] for the huge difference between a random system and the algebraic representation of cryptographic protocols). What we mean here is that if for some small number of rounds ElimLin performs well but then it stops working for more rounds, we can increase the number of samples and it will become effective again. The bottleneck is having an efficient data structure for implementing ElimLin together with a rigorous theory behind it to anticipate its behaviour. These two factors are currently missing in the literature.

Except two simple theorems by Bard (see Chapter 12, Section 5 of [4]), almost nothing has been done regarding the theory behind ElimLin. As ElimLin can also be used as a pre-processing step in any algebraic attack, building a proper theory is vital for improving the state of the art algebraic attacks. We are going to shed some lights on the way this ad-hoc algorithm works and the theory behind it.

In this paper, we show that the output of ElimLin is invariant with respect to any variable ordering. This is a surprising result, i.e., while the spaces generated are different depending on how substitution is performed, we prove that their intersection is exactly the same. Furthermore, we prove that no affine bijective variable change can modify the output of ElimLin. Then, we prove a theorem on how the number of linear equations evolves in each iteration of ElimLin.

An unannounced competition is currently running for designing lightweight cryptographic primitives. This includes several designs which have appeared in the last few years (see [7, 23, 40, 35, 30, 37, 47, 2, 36, 6]). These designs mainly compete over the gate equivalent (GE) and throughput. This might not be a fair comparison of efficiency, since they do not provide the same level of security with respect to distinct types of attacks. In this paper, we compare the two lightweight Feistel-based block ciphers MIBS [39] and LBlock [50] and show that with the same number of rounds, LBlock provides a much lower level of security compared to MIBS with respect to algebraic attacks. In fact, we attack both ciphers with ElimLin and F4 algorithm. Finally, we provide several algebraic attacks against Courtois Toy Cipher version 2 (CTC2) with distinct parameters using ElimLin.

In Sec. 2, we elaborate the ElimLin algorithm. Then, we remind some basic theorems on ElimLin in Sec. 3. As our main contribution (Theorem 7), we prove in Sec. 4 that ElimLin can be formulated as an intersection of vector spaces. We also discuss its consequences in Sec. 4.2 and prove a theorem regarding the evolution of linear equations in Sec. 4.3. We perform some attacks simulations on CTC2, LBlock and MIBS block ciphers in Sec. 5.2, 5.3 and 5.4 respectively. In Sec. 6, we compare ElimLin and F4. We mention some open problems and a conjecture in Sec. 7 and we conclude. Finally, in the Appendix we give a toy example on ElimLin and discuss the effect of multiple samples.

2 ElimLin Algorithm

ElimLin stands for **Elim**inate **Lin**ear and it is a technique for solving polynomial systems of multivariate equations of low degree d mostly: 2, 3, or 4 over a finite field specifically GF(2). It is also known as "inter-reduction" step in all major algebra systems. As a single tool, it was proposed in [17] to attack DES. It broke 5-round DES. Later, it was applied to break 5-round PRESENT block cipher [44] and to analyze the resistance of Snow 2.0 stream cipher against algebraic attacks [18]. It is a simple but a powerful algorithm which can be applied to any symmetric cipher and is capable of breaking their reduced versions. There is no specific requirement for the system except that there should exist at least one linear term, otherwise ElimLin trivially fails. The key question for such an algorithm is to predict its behavior. Currently, very similar to most other types of algebraic attacks such as [48, 24, 25], multiple parts of the algorithm are heuristic, so it is worthwhile to prove which factors can improve its results, make it run faster or does not have any influence on its ultimate result. This will yield a better understanding of how ElimLin works.
ElimLin is composed of two sequential distinct stages, namely:

- *Gaussian Elimination:* All the linear equations in the linear span of initial equations are found. They are the intersection between two vector spaces: The vector space spanned by all monomials of degree 1 and the vector space spanned by all equations.
- *Substitution:* Variables are iteratively eliminated in the whole system based on linear equations until there is no linear equation left. Consequently, the remaining system has fewer variables.

This routine is iterated until no linear equation is obtained in the linear span of the system. See Fig. 1 for a more precise definition of the algorithm. We also give a toy system of equations in the Appendix and solve it with ElimLin. Clearly, the algorithm shall depend on ordering strategies to apply in step 5, 11, and 12 of Fig. 1. We will see that it is not, i.e., the span of the resulting $\mathcal{S}_L$ is invariant.

We observe that new linear equations are derived in each iteration of the algorithm that did not exist in the former spans. This phenomenon is called *avalanche effect* in ElimLin and is the

consequence of Theorem 7. At the end, the system is solved linearly (when $\mathcal{S}_L$ is large enough) or ElimLin fails. If the latter occurs, we can increase the data complexity [3] and re-run the attack.

1: Input : A system of polynomial equations $\mathcal{S}^0 = \{\mathsf{Eq}_1^0, \ldots, \mathsf{Eq}_{m_0}^0\}$ over $\mathsf{GF}(2)$.
2: Output : An updated system of equations $\mathcal{S}^T$ and a system of linear equations $\mathcal{S}_L$.
3: Set $\mathcal{S}_L \leftarrow \emptyset$ and $\mathcal{S}^T \leftarrow \mathcal{S}^0$ and $k \leftarrow 1$.
4: **repeat**
5: Perform Gaussian elimination Gauss(.) on $\mathcal{S}^T$ with an arbitrary ordering of equations and monomials to eliminate non-linear monomials.
6: Set $\mathcal{S}_{L'} \leftarrow$ Linear equations from $\mathsf{Gauss}(\mathcal{S}^T)$.
7: Set $\mathcal{S}^T \leftarrow \mathsf{Gauss}(\mathcal{S}^T) \setminus \mathcal{S}_{L'}$.
8: Set flag.
9: **for** all $\ell \in \mathcal{S}_{L'}$ in an arbitrary order **do**
10: **if** ℓ is a trivial equation **then**
11: **if** ℓ is unsolvable **then**
12: Terminate and output "No Solution".
13: **end if**
14: **else**
15: Unset flag.
16: Let x_{t_k} be a monomial from ℓ.
17: Substitute x_{t_k} in $\mathcal{S}^T$ and $\mathcal{S}'_L$ using ℓ.
18: Insert ℓ in $\mathcal{S}_L$.
19: $k \leftarrow k + 1$
20: **end if**
21: **end for**
22: **until** flag is set.
23: Output $\mathcal{S}_T$ and $\mathcal{S}_L$.

Fig. 1. ElimLin algorithm.

3 State of the Art Theorems

The only theoretical analysis of ElimLin was done by Bard in [4]. He proved the following theorem and corollary for **one** iteration of ElimLin:

Theorem 1 ([4]). *All linear equations in the linear span of a polynomial equation system $\mathcal{S}^0$ are found in the linear span of linear equations derived by performing the first iteration of ElimLin algorithm on the system.*

The following corollary (also from [4]) is the direct consequence of the above theorem.

Corollary 2. *The linear equations generated after performing the first Gaussian elimination in ElimLin algorithm form a basis for all possible linear equations in the linear span of the system.*

This shows that any method to perform Gaussian elimination does not affect the linear space obtained at an arbitrary iteration of ElimLin. All linear equations derived from one method exist in the linear span of the equations cumulated from another method. This is trivial to see.

4 Algebraic Representation of ElimLin

4.1 ElimLin as an Intersection of Vector Spaces

We also formalize ElimLin in an algebraic way. This representation is used in proving Theorem 7. First, we define some notations.

[3] For instance, the number of plaintext-ciphertext pairs.

We call an *iteration* a Gaussian elimination preceding a substitution; The system of equations for ElimLin can be stored as a matrix $\mathcal{M}_\alpha$ of dimension $m_\alpha \times T_\alpha$, where each m_α rows represents an equation and each T_α columns represents a monomial at iteration α. Also, r_α denotes the rank of $\mathcal{M}_\alpha$. Let n_α be the number of variables at iteration α. We use a reverse lexicographical ordering of columns during Gaussian elimination to accumulate linear equations in the last rows of the matrix. Any arbitrary ordering can be used instead. In fact, we use the same matrix representation as described in [4].

Let $K = \mathsf{GF}(2)$ and $x = (x_1, \ldots, x_n)$ be a set of free variables. We denote by $K[x]$ the ring of multivariate polynomials over K. For $\mathcal{S} \subset K[x]$, we denote $\mathsf{Span}(\mathcal{S})$ the K-vector subspace of $K[x]$ spanned by $\mathcal{S}$. Let $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_n)$ be a power vector in $\mathbf{N}^n$. The term x^γ is defined as the product $x^\gamma = x_1^{\gamma_1} \times x_2^{\gamma_2} \times \cdots \times x_n^{\gamma_n}$. The total degree of x^γ is defined as $deg(x^\gamma) \stackrel{\mathsf{def}}{=} \gamma_1 + \gamma_2 + \cdots + \gamma_n$. Let $\mathsf{Ideal}(\mathcal{S})$ be the ideal spanned by $\mathcal{S}$ and $\mathsf{Root}(\mathcal{S})$ be the set of all tuples $m \in K^n$ such that $f(m) = 0$ for all $f \in \mathcal{S}$. Let

$$R_d = \mathsf{Span}\left(\text{monomials of degree} \leq d\right) / \mathsf{Ideal}\left(x_1^2 - x_1, x_2^2 - x_2, \ldots, x_n^2 - x_n\right)$$

Let $\mathcal{S}^\alpha$ be $\mathcal{S}^T$ after the α-th iteration of ElimLin and $\mathcal{S}^0$ be the initial system. Moreover, n_L^α is the number of non-trivial linear equations in $\mathcal{S}_{L'}$ at the α-th iteration. We denote $\mathcal{S}_L^\alpha$ the $\mathcal{S}_L$ after the α-th iteration. Also, $C^\alpha \stackrel{\mathsf{def}}{=} \#\mathcal{S}_L^\alpha$.

Let assume that $\mathcal{S}^0$ has degree bounded by d. We denote by $\mathsf{Var}(f)$ the set of variables x_i expressed in f. Let $x_{t_1}, \ldots, x_{t_k}$ be the sequence of eliminated variables. We define $\mathsf{V}_k = \{x_1, \ldots, x_n\} \backslash \{x_{t_1}, \ldots, x_{t_k}\}$. Also, let $\ell_1, \ell_2, \ldots, \ell_k$ be the sequence of linear equations as they are used during elimination (step 11 of Fig. 1). Hence, we have $x_{t_k} \in \mathsf{Var}(\ell_k) \subseteq \mathsf{V}_{k-1}$.

We prove the following crucial lemma which we use later to prove Theorem 7.

Lemma 3. *After the α-th iteration of ElimLin, an arbitrary equation Eq_i^α in the system $(\mathcal{S}^\alpha \cup \mathcal{S}_L^\alpha)$ for an arbitrary i can be represented as*

$$\mathsf{Eq}_i^\alpha = \sum_{t=1}^{m_0} \beta_{ti}^\alpha \cdot \mathsf{Eq}_t^0 + \sum_{t=1}^{C^\alpha} \ell_t(x) \cdot g_{ti}^\alpha(x) \tag{1}$$

where $\beta_{ti}^\alpha \in K$ and $g_{ti}^\alpha(x)$ is a polynomial in R_{d-1} and $\mathsf{Var}(g_{ti}^\alpha) \subseteq \mathsf{V}_t$.

Proof. Let x_{t_1} be one of the monomials existing in the first linear equation $\ell_1(x)$ and this specific variable is going to be eliminated. Substituting x_{t_1} in an equation $x_{t_1} \cdot h(x) + z(x)$, where $h(x)$ has degree at most $d-1$, $x_{t_1} \notin \mathsf{Var}(h)$ and $x_{t_1} \notin \mathsf{Var}(z)$ is identical to subtracting $h(x) \cdot \ell_1(x)$. Consequently, the proof follows by induction on α.

□

Now, we prove the inverse of the above lemma.

Lemma 4. *For each i and each α, there exists $\beta_{ti}'^\alpha \in K$ and $g_{ti}'^\alpha(x)$ such that*

$$\mathsf{Eq}_i^0 = \sum_{t=1}^{m_\alpha} \beta_{ti}'^\alpha \cdot \mathsf{Eq}_t^\alpha + \sum_{t=1}^{C^\alpha} \ell_t(x) \cdot g_{ti}'^\alpha(x) \tag{2}$$

where $g_{ti}'^\alpha(x)$ is a polynomial in R_{d-1} and $\mathsf{Var}(g_{ti}'^\alpha) \subseteq \mathsf{V}_t$.

Proof. Gaussian elimination and substitution are invertible operations. We can use a similar induction as the previous lemma to prove the above equation.

□

In the next lemma, we prove that $\mathcal{S}_L^\alpha$ contains all linear equations which can be written in the form of Eq. (1).

Lemma 5. *If there exists $\ell \in R_1$ and some β_t and $g_t''(x)$ such that*

$$\ell(x) = \sum_{t=1}^{m_0} \beta_t \cdot \mathsf{Eq}_t^0 + \sum_{t=1}^{C^\alpha} \ell_t(x) \cdot g_t''(x) \tag{3}$$

at iteration α, where $g_t''(x)$ is a polynomial in R_{d-1}, then there exists $u_t \in K$ and $v_t \in K$ such that

$$\ell(x) + \sum_{t=1}^{C^\alpha} u_t \cdot \ell_t(x) = \sum_{t=1}^{m_\alpha} v_t \cdot \mathsf{Eq}_t^\alpha$$

So, $\ell(x) \in \mathsf{Span}(\mathcal{S}_L^\alpha)$.

Proof. We define u_k iteratively: u_k is the coefficient of x_{t_k} in

$$\ell(x) + \sum_{t=1}^{k-1} u_t \cdot \ell_t(x)$$

for $k = 1, \ldots, C^\alpha$. So, $\mathsf{Var}(\ell(x) + \sum_{t=1}^{k} u_t \cdot \ell_t(x)) \subseteq \mathsf{V}_k$. By substituting Eq_i^0 from Eq. (2) in Eq. (3) and integrating u_t and g_t'' in $g_{ti}'^\alpha$, we obtain

$$\underbrace{\ell(x) + \sum_{t=1}^{C^\alpha} u_t \cdot \ell_t(x)}_{\subseteq \mathsf{V}_1} = \underbrace{\sum_{t=1}^{m_\alpha} v_t \cdot \mathsf{Eq}_t^\alpha}_{\subseteq \mathsf{V}_1} + \underbrace{\sum_{t=1}^{C^\alpha} \ell_t(x) \cdot g_t'(x)}_{\Longrightarrow \subseteq \mathsf{V}_1} \tag{4}$$

with $g_t'(x) \in R_{d-1}$. All $g_t'(x)$ where $t > 1$ can be written as $\bar{g}_t(x) + x_{t_1} \cdot \bar{\bar{g}}_t(x)$ with $\mathsf{Var}(\bar{g}_t) \subseteq \mathsf{V}_1$, $\mathsf{Var}(\bar{\bar{g}}_t) \subseteq \mathsf{V}_1$ and $\bar{\bar{g}}_t(x) \in R_{d-2}$. Since,

$$\begin{aligned} \ell_1(x) \cdot g_1'(x) + \ell_t(x) \cdot g_t'(x) = \ell_1(x) \cdot \underbrace{\left(g_1'(x) + \ell_t(x) \cdot \bar{\bar{g}}_t(x)\right)}_{\text{new } g_1'(x)} \\ + \underbrace{\ell_t(x)}_{\subseteq \mathsf{V}_1} \cdot \underbrace{\left(\bar{g}_t(x) + \bar{\bar{g}}_t(x) \cdot (x_{t_1} - \ell_1(x))\right)}_{(\text{new } g_t'(x)) \subseteq \mathsf{V}_1} \end{aligned}$$

we can re-arrange the sum in Eq. (4) using the above representation and obtain $\mathsf{Var}(g_t') \subseteq \mathsf{V}_1$ for all $t > 1$. Also, x_{t_1} only appears in $\ell_1(x)$ and $g_1'(x)$. So, the coefficient of x_{t_1} in the expansion of $\ell_1(x) \cdot g_1'(x)$ must be zero. In fact, we have

$$\begin{aligned} \ell_1(x) \cdot g_1'(x) &= (x_{t_1} + (\ell_1(x) - x_{t_1})) \cdot (\bar{g}_1(x) + x_{t_1} \cdot \bar{\bar{g}}_t(x)) \\ &= x_{t_1} \cdot (\bar{\bar{g}}_1(x) \cdot (1 + \ell_1(x) - x_{t_1}) + \bar{g}_1(x)) + \bar{g}_1(x) \cdot (\ell_1(x) - x_{t_1}) \end{aligned}$$

So, $\bar{g}_1(x) = \bar{\bar{g}}_1(x) \cdot (x_{t_1} - \ell_1(x) - 1)$ and we deduce,

$$g_1'(x) = \bar{\bar{g}}_1(x) \cdot (\ell_1(x) + 1)$$

over $\mathsf{GF}(2)$. But, then $\ell_1(x) \cdot g_1'(x) = 0$ over R, since $\ell_1(x) \cdot (\ell_1(x) + 1) = 0$. Finally, we iterate and obtain

$$\ell(x) + \sum_{t=1}^{C^\alpha} u_t \cdot \ell_t(x) = \sum_{t=1}^{m_\alpha} v_t \cdot \mathsf{Eq}_t^\alpha$$

□

1: Input : A set $\mathcal{S}^0$ of polynomial equations in R_d.
2: Output : A system of linear equations $\mathcal{S}_L$.
3: Set $\bar{\mathcal{S}_L} := \emptyset$.
4: **repeat**
5: $\quad \bar{\mathcal{S}_L} \leftarrow \mathsf{Span}\left(\mathcal{S}^0 \cup (R_{d-1} \times \bar{\mathcal{S}_L})\right) \cap R_1$
6: **until** $\bar{\mathcal{S}_L}$ unchanged
7: Output $\mathcal{S}_L$: a basis of $\bar{\mathcal{S}_L}$.

Fig. 2. ElimLin algorithm from another perspective.

From another perspective, ElimLin algorithm can be represented as in Fig. 2. In fact, as a consequence of Lemma 3 and Lemma 5, Fig. 2 presents a unique characterization of $\mathsf{Span}\,(\mathcal{S}_L)$ in terms of a fixed point:

Lemma 6. *At the end of ElimLin, $\mathsf{Span}\,(\mathcal{S}_L)$ is the smallest subset $\bar{\mathcal{S}_L}$ of R_1, such that*

$$\bar{\mathcal{S}_L} = \mathsf{Span}\left(\mathcal{S}^0 \cup (R_{d-1} \times \bar{\mathcal{S}_L})\right) \cap R_1$$

Proof. By induction, at step α we have $\bar{\mathcal{S}_L} \subseteq \mathsf{Span}\,(\mathcal{S}_L^\alpha)$, using Lemma 5. Also, $\mathcal{S}_L^\alpha \subseteq \bar{\mathcal{S}_L}$ using Lemma 3. So, $\bar{\mathcal{S}_L} = \mathsf{Span}\,(\mathcal{S}_L^\alpha)$ at step α. Since

$$\bar{\mathcal{S}_L} \mapsto \mathsf{Span}\left(\mathcal{S}^0 \cup (R_{d-1} \times \bar{\mathcal{S}_L})\right) \cap R_1$$

is increasing, we obtain the above equation.

□

ElimLin eliminates variables, thus it looks very unexpected that the number of linear equations in each step of the algorithm is invariant with respect to any variable ordering in the substitution step and the Gaussian elimination. We finally prove this important invariant property. Concretely, we formalize ElimLin as a sequence of intersection of vector spaces. Such intersection in each iteration is between the vector space spanned by the equations and the vector space generated by all monomials of degree 1 in the system. This implies that if ElimLin runs for α iterations (finally succeeds or fails), it can be formalized as a sequence of intersections of α pairs of vector spaces. These intersections of vector spaces only depend on the vector space of the initial system.

Theorem 7. *The following relations exist after running ElimLin on a polynomial system of equations Q:*

1. $\mathsf{Root}\left(\mathcal{S}^0\right) = \mathsf{Root}(\mathcal{S}^T \cup \mathcal{S}_L)$
2. *There is no linear equation in* $\mathsf{Span}\left(\mathcal{S}^T\right)$.
3. $\mathsf{Span}\,(\mathcal{S}_L)$ *is uniquely defined by* $\mathcal{S}^0$.
4. $\mathcal{S}_L$ *consists of linearly independent linear equations.*
5. *The complexity is* $O\left(n_0^{d+1} m_0^2\right)$, *where d is the degree of the system and n_0 and m_0 are the initial number of variables and equations, respectively.*

Proof (1). Due to Lemma 3 and Lemma 4, $\mathcal{S}^0$ and $(\mathcal{S}^T \cup \mathcal{S}_L)$ are equivalent. So, a solution of $\mathcal{S}^0$ is also a solution of $(\mathcal{S}^T \cup \mathcal{S}_L)$ and vice versa.

Proof (2). Since ElimLin stops on $\mathcal{S}^T$, the Gaussian reduction did not find any linear polynomial.

Proof (3). Due to Lemma 6.

Proof (4). $\mathcal{S}_L$ includes a basis for $\bar{\mathcal{S}_L}$. So, it consists of linearly independent equations.

Proof (5). n_0 is an upper bound on $\#\mathcal{S}_L$ due to the fact that $\mathcal{S}_L$ consists of linearly independent linear equations. So, the number of iterations is bounded by n_0. The total number of monomials is bounded by

$$T_0 \leq \sum_{i=0}^{d} \binom{n_0}{i} = O\left(n_0^d\right)$$

The complexity of Gaussian elimination is $O(m_0^2 T_0)$, since we have T_0 columns and m_0 equations. Therefore, overall, the complexity of ElimLin is $O\left(n_0^{d+1} m_0^2\right)$.

□

4.2 Affine Bijective Variable Change

In the next theorem, we prove that the result of ElimLin algorithm does not change for any affine bijective variable change. It is an open problem to find an appropriate non-linear variable change which improves the result of the ElimLin algorithm.

Theorem 8. *Any affine bijective variable change* $A : \mathsf{GF}(2)^{n_0} \to \mathsf{GF}(2)^{n_0}$ *on a* n_0*-variable system of equations* $\mathcal{S}^0$ *does not affect the result of* ElimLin *algorithm, implying that the number of linear equations generated at each iteration is invariant with respect to an affine bijective variable change.*

Proof. In Lemma 6, we showed that $\mathsf{Span}\,(\mathcal{S}_L)$ is the output of the algorithm in Fig. 2, iterating

$$\bar{\mathcal{S}_L} \leftarrow \mathsf{Span}\left(\mathcal{S}^0 \cup (R_{d-1} \times \bar{\mathcal{S}_L})\right) \cap R_1$$

We represent the composition of a polynomial f_1 with respect to A by $\mathsf{Com}(f_1)$. We then show that there is a commutative diagram

$$\begin{array}{ccc} \mathcal{S}^0 & \xrightarrow{\mathsf{Com}} & \mathsf{Com}(\mathcal{S}^0) \\ \Big\downarrow{\scriptstyle \mathsf{ElimLin}} & & \Big\downarrow{\scriptstyle \mathsf{ElimLin}} \\ \bar{\mathcal{S}_L} & \xrightarrow[\mathsf{Com}]{} & \mathsf{Com}(\bar{\mathcal{S}_L}) \end{array}$$

We consider two parallel executions of the algorithm in Fig. 2, one with $\mathcal{S}^0$ and the other with $\mathsf{Com}(\mathcal{S}^0)$.
If we compose the polynomials in $\mathcal{S}^0$ with respect to A, in the above relation R_{d-1} remains the same. Since the transformation A is affine,

$$\mathsf{Com}(\mathsf{Span}\left(\mathcal{S}^0 \cup (R_{d-1} \times \bar{\mathcal{S}_L})\right) \cap R_1) = \mathsf{Span}\left(\mathsf{Com}(\mathcal{S}^0) \cup (R_{d-1} \times \mathsf{Com}(\bar{\mathcal{S}_L}))\right) \cap R_1$$

So, at each iteration, the second execution has the result of applying Com to the result of the first one.

□

4.3 Linear Equations Evolution

An open problem regarding ElimLin is to predict how the number of linear equations evolves in the preceding iterations. In the following theorem, we give a necessary (but not sufficient) condition for a dense overdefined system of equations to have an additional linear equation in the next iteration of ElimLin. Proving a similar result for a sparse system is not straightforward.

Theorem 9. *If we apply ElimLin to an overdefined dense system of quadratic equations over* GF(2)*, for* $n_L^{\alpha+1} > n_L^{\alpha}$ *to hold, it is necessary to have*

$$\frac{b_\alpha}{2} - a_\alpha < n_L^\alpha < \frac{b_\alpha}{2} + a_\alpha$$

where $b_\alpha = 2n_\alpha - 1$ *and* $a_\alpha = \frac{\sqrt{b_\alpha^2 - 8n_L^\alpha}}{2}$.

Proof. For the system to generate linear equations, it is necessary that the *sufficient rank condition* [4] is satisfied. More clearly, we must have $r_\alpha > T_\alpha - 1 - n_\alpha$, otherwise no linear equations will be generated. This is true if the system of equations is overdefined. Hence, we obtain,

$$n_L^\alpha = r_\alpha + n_\alpha + 1 - T_\alpha \tag{5}$$

If some columns of the matrix $\mathcal{M}_\alpha$ are pivotless, it will shift the diagonal strand of ones to the right. Therefore, n_L^α will be more than what the above equation expresses. Assuming the system of equations is dense, this phenomenon happens with a very low probability. Suppose the above equation is true with high probability, then we get

$$n_L^{\alpha+1} = r_{\alpha+1} + n_{\alpha+1} + 1 - T_{\alpha+1} \tag{6}$$

In the $(\alpha+1)$-th iteration, the number of variables is reduced by n_L^α. Thus, $n_{\alpha+1} = n_\alpha - n_L^\alpha$. If the system of equations is dense, in a quadratic system,

$$T_\alpha = \binom{n_\alpha}{2} + n_\alpha + 1$$

and so,

$$T_{\alpha+1} = \binom{n_\alpha - n_L^\alpha}{2} + n_\alpha - n_L^\alpha + 1$$

Consequently, we have

$$T_\alpha - T_{\alpha+1} = n_L^\alpha \left(n_\alpha - \frac{1}{2}(n_L^\alpha - 1) \right) \tag{7}$$

Therefore, using Eq. (5), Eq. (6) and Eq. (7), we obtain,

$$\begin{aligned} n_L^{\alpha+1} &= (r_{\alpha+1} - r_\alpha) + (r_\alpha + n_\alpha - T_\alpha + 1) + n_L^\alpha(-\tfrac{1}{2}n_L^\alpha + n_\alpha - \tfrac{1}{2}) \\ &= n_L^\alpha(-\tfrac{1}{2}n_L^\alpha + n_\alpha + \tfrac{1}{2}) - (r_\alpha - r_{\alpha+1}) \end{aligned}$$

If $n_L^{\alpha+1} > n_L^\alpha$, then $n_L^\alpha(-\frac{1}{2}n_L^\alpha + n_\alpha + \frac{1}{2}) - (r_\alpha - r_{\alpha+1}) > n_L^\alpha$ and this leads to

$$n_L^{\alpha 2} + (1 - 2n_\alpha)n_L^\alpha + 2(r_\alpha - r_{\alpha+1}) < 0$$

$\Delta = (1 - 2n_\alpha)^2 - 8(r_\alpha - r_{\alpha+1})$, and if the above inequality holds, Δ should be positive and assuming $b_\alpha = 2n_\alpha - 1$, then, $b_\alpha - \sqrt{\Delta} < 2n_L^\alpha < b_\alpha + \sqrt{\Delta}$.

Considering Δ is positive, $n_\alpha > \sqrt{2(r_\alpha - r_{\alpha+1})} + \frac{1}{2}$. We also know that $r_{\alpha+1} \leq r_\alpha - n_L^\alpha$, which together lead to $n_\alpha > \frac{1}{2} + \sqrt{2n_L^\alpha}$. Therefore, for $n_L^{\alpha+1} > n_L^\alpha$, it is necessary to have $n_\alpha > \frac{1}{2} + \sqrt{2n_L^\alpha}$, but not visa versa. Simplifying $b_\alpha - \sqrt{\Delta} < 2n_L^\alpha < b_\alpha + \sqrt{\Delta}$ and deploying $r_\alpha - r_{\alpha+1} \geq n_L^\alpha$ results in

$$b_\alpha - 2a_\alpha < 2n_L^\alpha < b_\alpha + 2a_\alpha$$

where $b_\alpha = 2n_\alpha - 1$ and $2a_\alpha = \sqrt{b_\alpha^2 - 8n_L^\alpha}$.

Notice that $n_\alpha > \frac{1}{2} + \sqrt{2n_L^\alpha}$, which was obtained in the first stage of the proof, has been originated from the fact that $b_\alpha^2 - 8n_L^\alpha$ should be non-negative.

□

5 Attacks Simulations

In this section, we present our experimental results against CTC2, LBlock and MIBS block ciphers. The simulations for CTC2 were run on an ordinary PC with a 1.8 Ghz CPU and 2 GB RAM. All the other simulations were run on an ordinary PC with a 2.8 Ghz CPU and 4 GB RAM. The amount of RAM required by our implementation is negligible.

In our attacks, we build a system of quadratic equations with variables representing plaintext, ciphertext, key and state bits, which allows to express the system of equations of high degree as quadratic equations. Afterwards, for each sample we set the plaintext and ciphertext according to the result of the input/output of the cipher. In order to test the efficiency of the algebraic attack, we guess some bits of the key and set the key variables corresponding to the guess. Then, we run the solver (ElimLin, F4 or SAT solver) to recover the remaining key bits and test whether the guess was correct. Therefore, the complexity of our algebraic attack can be bounded by $2^g \cdot \mathcal{C}(solver)$, where $\mathcal{C}(solver)$ represents the running time of the solver and g is the number of bits we guess. $\mathcal{C}(solver)$ is represented as the the *"Running Time"* in all the following tables.

For a comparison with a brute force attack, we consider a fair implementation of the cipher, which requires 10 CPU cycles per round. This implies that the algebraic attack against t rounds of the cipher is faster than an exhaustive search for the 1.8 Ghz and 2.8 Ghz CPU iff recovering c bits of the key is faster than $5.55 \cdot t \cdot 2^{c-31}$ and $3.57 \cdot t \cdot 2^{c-31}$ seconds respectively. This is already twice faster than the complexity of exhaustive search. All the attacks reported in the following tables are faster than exhaustive search with the former argument. In fact, we consider the cipher to be broken for some number of rounds if the algebraic attack that recovers $(\#key - g)$ key bits is faster than an exhaustive key search over $(\#key - g)$ bits of the key.

5.1 Simulations Using F4 Algorithm under PolyBoRi Framework

The most efficient implementation of the F4 algorithm is available under PolyBoRi framework [9] running alone or under SAGE algebra system. PolyBoRi is a C++ library designed to compute Gröbner basis of an ideal applied to Boolean polynomials. A Python interface is used, surrounding the C++ core. It uses zero-suppressed binary decision diagrams (ZDDs) [34] as a high level data structure for storing Boolean polynomials. This representation stores the monomials more efficiently in memory and it makes the Gröbner basis computation faster compared to other algebra systems.

We use polybori-0.8.0 for our attacks. Together with ElimLin, we also attack LBlock and MIBS with F4 algorithm and then compare PolyBoRi's efficiency with our implementation of ElimLin.

5.2 Simulations on CTC2

Courtois Toy Cipher (CTC) is an SPN-based block cipher devised by Courtois [14] as a toy cipher to evaluate algebraic attacks on smaller variants of cryptosystems. It was designed to show that it is possible to break a cipher using an ordinary PC deploying a small number of known or chosen plaintext-ciphertext pairs.

Since the system of equations of well-known ciphers such as AES is often large, it is not feasible by the current algorithms and computer capacities to solve them in a reasonable time, therefore smaller but similar versions such as CTC can be exploited to evaluate the resistance of ciphers against algebraic cryptanalysis. This turns out to yield a benchmark on understanding the algebraic structure of ciphers. Ultimately, this might lead to break of a larger system later.

CTC was not designed to be resistant against all known types of attacks like linear and differential cryptanalysis. Nevertheless, in [26], it was attacked by linear cryptanalysis. Subsequently, CTC Version 2 or CTC2 was proposed [13] to resolve the flaw exists in CTC structure. CTC2 is very similar to CTC with a few changes. It is an SPN-based network with scalable number of rounds, block and key size. For the full specification, refer to [13]. In CT-RSA 2009, differential and differential-linear attacks could reach up to 8 rounds of CTC2 [27], but as stated before, the objective of the CTC designer was not applying statistical attacks to his design. Finally, there is a cube attack on 4 rounds of one variant of this cipher in [42].

Since block size and key size are flexible in CTC2 cipher, we break various versions with distinct parameters (see Table 1) using ElimLin. The block size is specified by a parameter B, which specifies the number of parallel S-boxes per round. CTC2 S-box is 3×3, hence the block size is computed as $3B$. We guess some LSB bits of the key and we show that recovering the remaining is faster than exhaustive search.

It might be possible that during the intermediate steps of ElimLin, a quadratic equation in only key bits (possibly linear) appears. In such cases, approximately $\mathcal{O}(\#key^2)$ samples are enough to break the system. This is due to the fact that we can simply change the plaintext-ciphertext pair and generate a new linearly independent equation in the key. Finally, when we have enough such equations, we solve a system of quadratic equations in only key bits using the linearization technique. When such phenomenon occurs, intuitively the cipher is close to be broken but not yet. We can increase the number of samples and most often it makes the cipher thoroughly collapse.

5.3 Simulations on LBlock

LBlock is a new lightweight Feistel-based block cipher, aimed at constrained environments, such as RFID tags and sensor networks [50] proposed at ACNS 2011. It operates on 64-bit blocks, uses a key of 80 bits and iterates 32 rounds. For a detailed specification of the cipher, refer to [50]. As far as the authors are aware, there is currently no cryptanalysis results published on this cipher.

We break 8 rounds of LBlock using 6 samples deploying an ordinary PC by ElimLin. Our results are summarized in Table 2. In the same scenario, PolyBoRi crashes due to running out of memory.

5.4 Simulations on MIBS

Similar to the LBlock block cipher, MIBS is also a lightweight Feistel-based block cipher, aimed at constrained environments, such as RFID tags and sensor networks [39]. It operates on 64-bit blocks, uses keys of 64 or 80 bits and iterates 32 rounds. For a detailed specification of the cipher, see [39].

Table 1. CTC2 simulations using ElimLin up to 6 rounds with distinct parameters.

B	N_r	$\#key$	g	Running Time[1] (in hours)	Running Time[2] (in hours)	Data	Attack notes
16	3	48	0	0.03		5 KP	ElimLin
16	3	48	0		0.12	14 KP	ElimLin
64	3	192	155		0.03	1 KP	ElimLin
85	3	255	210		0.04	1 KP	ElimLin
16	4	48	0	0.01		2 CP	ElimLin
16	4	48	0		0.05	4 CP	ElimLin
40	4	120	85	0.00		1 KP	ElimLin
40	4	120	85		0.84	16 KP	ElimLin
48	4	144	100		0.12	4 KP	ElimLin
64	4	192	148	0.05		1 KP	ElimLin
64	4	192	155		2.21	5 KP	ElimLin
85	4	255	220	0.29		1 KP	ElimLin
85	4	255	215	0.64		1 KP	ElimLin
85	4	255	220		0.26	2 KP	ElimLin
85	4	255	215		0.90	3 KP	ElimLin
85	4	255	210		1.33	4 KP	ElimLin
16	5	48	0	3		8 CP	ElimLin
40	5	120	85		0.03	2 CP	ElimLin
32	6	96	60	2.5		16 CP	ElimLin
40	6	120	80	1		8 CP	ElimLin
64	6	192	155	2.4		4 CP	ElimLin
85	6	255	210	3		2 CP	ElimLin
85	6	255	220		3	16 CP	ElimLin
85	6	255	210		180.5	64 CP	ElimLin
128	6	384	344		4.5	2 CP	ElimLin

B : Number of S-boxes per round. To obtain the block size, B is multiplied by 3.
N_r : Number of rounds
g: Number of guessed LSB of the key
Running Time[1]: Running time until we achieve equations only in key variables (no other internal variables). When this is achieved, the cipher is close to be broken, but not yet (see Sec. 5.2).
Running Time[2]: Attack running time for recovering $(\#key - g)$ bits of the key.
KP: Known plaintext
CP: Chosen plaintext

Table 2. Algebraic attack complexities on reduced-round LBlock using ElimLin and PolyBoRi.

N_r	$\#key$	g	Running Time (in hours)	Data	Attack notes
8	80	32	0.252	6 KP	ElimLin
8	80	32	crashed	6 KP	PolyBoRi

N_r : Number of rounds
g: Number of guessed LSB of the key
KP: Known plaintext
CP: Chosen plaintext

Currently, the best cryptanalysis results is a linear attack reaching 18-round MIBS with data complexity 2^{61} and time complexity of 2^{76} [5]. In fact, statistical attacks often require very large number of samples. This is not always achievable in practice.

We break 4 and 3 rounds of MIBS80 and MIBS64 using 32 and 2 samples deploying an ordinary PC by ElimLin. Our results are summarized in Table 3. In 2 out of 3 experiments, PolyBoRi crashes due to running out of memory. This is the first algebraic analysis of the cipher.

The designers in [39] have evaluated the security of their cipher with respect to algebraic attacks. They used the complexity of XSL algorithm for this evaluation, which is not a precise measurement for evaluating resistance of a cipher against algebraic attacks, since effectiveness of XSL is still controversial and under speculation. There are better methods such as SAT solvers [3] which solve MQ problem faster than expected due to the system being overdefined and sparse.

Let assume XSL can be precise enough to evaluate the security of a cipher with respect to algebraic attacks. According to [21, 39], the complexity of XSL can be evaluated with the work factor. For MIBS, work factor is computed as:

$$\mathsf{WF} = \Gamma^{\omega} \left((\text{Block Size}) \cdot N_r^2 \right)^{\omega \lceil \frac{T}{r} \rceil}$$

where Γ is a parameter which depends only on the S-box. For MIBS, $\Gamma = 85.56$. The value $r = 21$ is the number of equations the S-box can be represented with. $T = 37$ is the number of monomials in that representation. $\omega = 2.37$ is the exponent of the Gaussian elimination complexity. The work factor for attacking 5-round MIBS is $\mathsf{WF} = 2^{65.65}$ which is worse than an exhaustive key search for MIBS64. Deploying SAT solving techniques using MiniSAT 2.0 [29], we can break 5 rounds of MIBS64 (see Table 3). Our strategy is exactly the same as [3]. Table 3 already shows that we can do better than $2^{65.65}$ for MIBS64. We can perform a very similar attack on MIBS80. This already shows that considering the complexity of XSL is not a precise measure to evaluate the security of a cipher against algebraic cryptanalysis. Complexity of attacking such system with XL is extremely high.

We believe that due to the similarity between the structure of MIBS and LBlock, we can compare them with respect to algebraic attacks. As can be seen from the table of attacks, LBlock is much weaker. This is not surprising though, since the linear layer of LBlock is much weaker than MIBS, since it is nibble-wise instead of bit-wise. So, we could attack twice more rounds of LBlock. Thus, although LBlock is lighter with respect to the number of gates, but it provides a lower level of security with respect to algebraic attacks.

Table 3. Algebraic attack complexities on reduced-round MIBS using ElimLin, PolyBoRi and MiniSAT 2.0.

N_r	$\#key$	g	Running Time (in hours)	Data	Attack notes
4	80	20	0.137	32 KP	ElimLin
4	80	20	crashed	32 KP	PolyBoRi
5	64	16	0.395	6 KP	MiniSAT 2.0
5	64	16	crashed	6 KP	PolyBoRi
3	64	0	0.006	2 KP	ElimLin
3	64	0	0.002	2 KP	PolyBoRi

N_r : Number of rounds
g: Number of guessed LSB of the key
KP: Known plaintext
CP: Chosen plaintext

6 A Comparison Between ElimLin and PolyBoRi

Gröbner basis is currently one of the most successful methods for solving polynomial systems of equations. However, it has its own restrictions. The main bottleneck of the Gröbner basis techniques is the memory requirement and therefore most of the Gröbner basis attacks use relatively small number of samples. It is worthwhile to mention that ElimLin is a subroutine is

Gröbner basis computations. But, ElimLin algorithm as a single tool requires a large number of samples to work.

The Gröbner basis solve the system by reductions according to a pre-selected ordering, which can lead to high degree dense polynomials. ElimLin uses the fact that multiple samples provide an additional information to the solver, and therefore the key might be found even if when we restrict the reduction to degree 2.

Next, we compare the current state of the art implementation of F4 algorithm PolyBoRi and our implementation of ElimLin. In the cases where ElimLin behaves better than PolyBoRi, it does not mean that ElimLin is superior to F4 algorithm. In fact, it just means that there exists a better implementation for ElimLin than for F4 for some particular systems of equations. F4 uses a fixed ordering for monomials and therefore it does not preserve the sparsity in its intermediate steps. On the other hand, our implementation of ElimLin performs several sparsity preserving techniques by changing the ordering. This drops the total number of monomials and makes it memory efficient.

Table 2 and Table 3 show that PolyBoRi requires too much memory and crashes for a large number of samples. At the same time, our implementation of ElimLin is slightly slower than PolyBoRi implementation attacking 2 samples of 3-round MIBS64 as in Table 3. This demonstrates that our implementation of ElimLin can be more effective than PolyBoRi and vice versa, depending on memory requirements of PolyBoRi. However, whenever the system is solvable by our implementation of ElimLin, our experiments revealed that PolyBoRi does not give a significant advantage over ElimLin because the memory requirements are too high.

While PolyBoRi may yield a solution for a few samples, the success of ElimLin is determined by the number of samples provided to the algorithm. The evaluation of the number of sufficient samples in ElimLin is still an open problem.

We see that often preserving the degree by simple linear algebra techniques can outperform the more sophisticated Gröbner basis algorithms, mainly due to the structural properties of the system of equations of a cryptographic primitive (such as sparsity). ElimLin takes advantage of such structural properties and uncovers some hidden linear equations using multiple samples. According to our experiments, PolyBoRi does not seem to be able to take advantage of these structural properties as would be expected which results in higher memory requirements than would be necessary and ultimately their failure for large systems, even though it is clearly possible for the algorithm to find the solution in reasonable time. Finally, we need more efficient implementations and data structures for both ElimLin and Gröbner basis algorithms.

7 Further Work and Some Conjectures

An interesting area of research is to estimate the number of linear equations in ElimLin or anticipate how this number evolves in the succeeding iterations or evaluate after how many iterations ElimLin finishes. Also, to anticipate how many samples is enough to make the system collapse by ElimLin. Last but not least, it is prominent to find a very efficient method for implementing ElimLin and to find the most appropriate data structure to choose.

There are some evidence which illustrate that ElimLin does not reveal all hidden linear equations in the structure of the cipher up to a specific degree. We give an example, demonstrating such an evidence:

Assume there exists an equation in the system which can be represented as $\ell(x)g(x)+1=0$ over $\mathsf{GF}(2)$, where $\ell(x)$ is a polynomial of degree one and $g(x)$ is a polynomial of degree at most $d-1$. Running ElimLin on this single equation trivially fails. But, if we multiply both sides of the equation by $\ell(x)$, we obtain $\ell(x)g(x)+\ell(x)=0$. Summing these two equations, we derive $\ell(x)=1$. This hidden linear equation can be simply captured by the XL algorithm, but can not

be captured by ElimLin. There exist multiple other examples which demonstrate that ElimLin does not generate all the hidden linear equations. To generate all such linear equations, the degree-bounded Gröbner basis can be used.

For big ciphers, for example the full AES, it is also plausible that:

Conjecture 1 *For each number of rounds X, there exists Y such that AES is broken by ElimLin given Y Chosen or Known Plaintext-Ciphertext pairs.*

Disproving the above conjecture leads to the statement that "AES can not be broken by algebraic attack at degree 2". But maybe this conjecture is true, then the capacities of the ElimLin attack are considerable and it works for any number of rounds X. As a consequence, if for $X = 14$ this Y is not too large, say less than 2^{64}, the AES-256 will be broken faster than brute force by ElimLin at degree 2, which is much simpler than Gröbner basis objective of breaking it at degree 3 or 4 with 1 KP.

ElimLin is a polynomial time algorithm. If it can be shown that a polynomial number of samples is enough to gain a high success rate for ElimLin, this can already be considered a breakthrough in cryptography. Unfortunately, the correctness of this statement is not clear.

Conclusion

In this paper, we proved that ElimLin can be formulated in terms of a sequence of intersections of vector spaces. We showed that different monomial orderings and any affine bijective variable change do not influence the result of the algorithm. We did some predictions on the evolution of linear equations in the succeeding iterations in ElimLin. We presented multiple attacks deploying ElimLin against CTC2, LBlock and MIBS block ciphers.

References

1. F. Armknecht and G. Ars. Algebraic Attacks on Stream Ciphers with Gröbner Bases. *Gröbner Bases, Coding, and Cryptography*, pages 329–348, 2009.
2. J. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia. Quark: A Lightweight Hash. In *CHES*, volume 6225, pages 1–15, 2010.
3. G. Bard, N. Courtois, and C. Jefferson. Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers. In *presented at ECRYPT workshop Tools for Cryptanalysis*, 2007. http://eprint.iacr.org/2007/024.pdf.
4. G.V. Bard. *Algebraic Cryptanalysis*. Springer, 2009.
5. A. Bay, J. Nakahara, and S. Vaudenay. Cryptanalysis of Reduced-Round MIBS Block Cipher. In *CANS*, volume 6467, pages 1–19. Springer, 2010.
6. A. Bogdanov, M. Knežević, G. Leander, D. Toz, K. Varici, and I. Verbauwhede. SPONGENT: A Lightweight Hash Function. In *CHES*, volume 6917, pages 312–325, 2011.
7. A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *CHES*, volume 4727, pages 450–466, 2007.
8. M. Brickenstein and A. Dreyer. PolyBoRi: A framework for Gröbner basis computations with Boolean polynomials. In *Electronic Proceedings of MEGA 2007*, 2007. http://www.ricam.oeaw.ac.at/mega2007/electronic/26.pdf.
9. M. Brickenstein and A. Dreyer. PolyBoRi: A framework for Gröbner basis computations with Boolean polynomials. In *Electronic Proceedings of MEGA*, 2007.
10. B. Buchberger. Bruno Buchberger's PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. *Journal of Symbolic Computation*, 41(3-4):475–511, 2006.
11. N. Courtois. Higher Order Correlation Attacks, XL Algorithm and Cryptanalysis of Toyocrypt. In *ICISC*, volume 2587, pages 182–199, 2002.
12. N. Courtois. Fast Algebraic Attacks on Stream Ciphers with Linear Feedback. In *Advances in Cryptology - CRYPTO*, volume 2729, pages 176–194. Springer, 2003.

13. N. Courtois. CTC2 and Fast Algebraic Attacks on Block Ciphers Revisited. In *Cryptology ePrint Archive*, 2007. http://eprint.iacr.org/2007/152.pdf.
14. N. Courtois. How Fast can be Algebraic Attacks on Block Ciphers? In *Symmetric Cryptography*, volume 07021 of *Dagstuhl Seminar Proceedings*, 2007.
15. N. Courtois. The Dark Side of Security by Obscurity - and Cloning MiFare Classic Rail and Building Passes, Anywhere, Anytime. In *SECRYPT*, pages 331–338, 2009.
16. N. Courtois. Algebraic Complexity Reduction and Cryptanalysis of GOST. In *Cryptology ePrint Archive*, 2011. http://eprint.iacr.org/2011/626.
17. N. Courtois and G.V. Bard. Algebraic Cryptanalysis of the Data Encryption Standard. In *IMA Int. Conf.*, volume 4887, pages 152–169. Springer, 2007.
18. N. Courtois and B. Debraize. Algebraic Description and Simultaneous Linear Approximations of Addition in Snow 2.0. In *ICICS*, volume 5308, pages 328–344. Springer, 2008.
19. N. Courtois and W. Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In *Advances in Cryptology - EUROCRYPT*, volume 2656, pages 345–359. Springer, 2003.
20. N. Courtois, S. O'Neil, and J. Quisquater. Practical Algebraic Attacks on the Hitag2 Stream Cipher. In *ISC*, volume 5735, pages 167–176, 2009.
21. N. Courtois and J. Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In *Asiacrypt*, volume 2501, pages 267–287. Springer, 2002.
22. N. Courtois, A. Shamir, J. Patarin, and A. Klimov. Efficient Algorithms for Solving Overfined Systems of Multivariate Polynomial Equations. In *Advances in Cryptology, Eurocrypt*, volume 1807, pages 392–407. Springer, 2000.
23. C. De Canniére, O. Dunkelman, and M. Knezević. KATAN and KTANTAN - a family of small and efficient hardware-oriented block ciphers. In *CHES*, volume 5747, pages 272–288, 2009.
24. I. Dinur and A. Shamir. Cube Attacks on Tweakable Black Box Polynomials. In *Advances in Cryptology - EUROCRYPT*, volume 5479, pages 278–299. Springer, 2009.
25. I. Dinur and A. Shamir. Breaking Grain-128 with Dynamic Cube Attacks. In *FSE*, volume 6733, pages 167–187. Springer, 2011.
26. O. Dunkelman and N. Keller. Linear Cryptanalysis of CTC. In *Cryptology ePrint Archive*, 2006. http://eprint.iacr.org/2006/250.pdf.
27. O. Dunkelman and N. Keller. Cryptanalysis of CTC2. In *CT-RSA*, volume 5473, pages 226–239. Springer, 2009.
28. N. Eén and N. Sörensson. MiniSat 2.0. An open-source SAT solver package. http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/.
29. N. Een and N. Sorensson. Minisat - A SAT Solver with Conflict-Clause Minimization. In *Theory and Applications of Satisfiability Testing*, 2005.
30. D. Engels, M.O. Saarinen, P. Schweitzer, and E.M. Smith. The Hummingbird-2 Lightweight Authenticated Encryption Algorithm. In *RFIDsec*, volume 7055, pages 19–31, 2011.
31. J. Faugère. A new effcient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1-3):61–88, 1999.
32. J. Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *Symbolic and Algebraic Computation - ISSAC*, page 7583, 2002.
33. G. Fusco and E. Bach. Phase transition of multivariate polynomial systems. *Journal of Mathematical Structures in Computer Science*, 19(1), 2009.
34. M. Ghasemzadeh. *A New Algorithm for the Quantified Satisfiability Problem, Based on Zero-suppressed Binary Decision Diagrams and Memoization.* PhD thesis, University of Potsdam, Germany, 2005.
35. Z. Gong, S. Nikova, and Y.W. Law. KLEIN: A New Family of Lightweight Block Ciphers. In *RFIDsec*, volume 7055, pages 1–18, 2011.
36. J. Guo, T. Peyrin, and A. Poschmann. The PHOTON Family of Lightweight Hash Functions. In *CRYPTO*, volume 6841, pages 222–239, 2011.
37. J. Guo, T. Peyrin, A. Poschmann, and M.J.B Robshaw. The LED Block Cipher. In *CHES*, volume 6917, pages 326–341, 2011.
38. S. Indesteege, N. Keller, O. Dunkelman, E. Biham, and B. Preneel. A Practical Attack on Keeloq. In *EUROCRYPT*, volume 4965, pages 1–18, 2008.
39. M. Izadi, B. Sadeghiyan, S. Sadeghian, and H. Arabnezhad. MIBS: A New Lightweight Block Cipher. In *CANS*, volume 5888, pages 334–348. Springer, 2009.
40. L.R. Knudsen, G. Leander, A. Poschmann, and M.J.B. Robshaw. PRINTcipher: A Block Cipher for IC-Printing. In *CHES*, volume 6225, pages 16–32, 2010.
41. Magma, software package. http://magma.maths.usyd.edu.au/magma/.
42. P. Mroczkowski and J. Szmidt. The Cube Attack on Courtois Toy Cipher. In *Cryptology ePrint Archive*, 2009. eprint.iacr.org/2009/497.pdf.
43. S. Murphy and M. Robshaw. Essential Algebraic Structure within AES. In *Advances in Cryptology - CRYPTO*, volume 2442, pages 1–16. Springer-Verlag, 2002.

44. J. Nakahara, P. Sepehrdad, B. Zhang, and M. Wang. Linear (Hull) and Algebraic Cryptanalysis of the Block Cipher PRESENT. In *CANS*, volume 5888, pages 58–75. Springer, 2009.
45. H. Raddum and I. Semaev. Solving Multiple Right Hand Sides linear equations. *Journal of Designs, Codes and Cryptography*, 49(1-3):147–160, 2008.
46. C.E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28, 1949.
47. K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai. Piccolo: An Ultra-Lightweight Blockcipher. In *CHES*, volume 6917, pages 342–357, 2011.
48. M. Vielhaber. Breaking ONE.FIVIUM by AIDA an Algebraic IV Differential Attack. In *Cryptology ePrint Archive*, 2007. http://eprint.iacr.org/2007/413.
49. R. Weinmann. Evaluating Algebraic Attacks on the AES. Master's thesis, Technische Universität Darmstadt, 2003.
50. W. Wu and L. Zhang. LBlock: A Lightweight Block Cipher. In *ACNS*, volume 6715, pages 327–344. Springer, 2011.

A A Toy Example of ElimLin

Let assume that we have the following overdefined system of multivariate equations over $\mathsf{GF}(2)$ with 5 variables $x_1, \ldots, x_5$ and 6 equations,

$$\begin{cases} x_1x_2 + x_1x_3 + x_2x_5 + x_3x_5 + x_2 + x_4 + x_5 + 1 = 0 \\ x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 = 0 \\ x_1x_4 + x_2x_3 + x_3 + 1 = 0 \\ x_1x_4 + x_1x_5 + x_2x_5 + x_1 = 0 \\ x_1x_5 + x_2x_3 + x_3x_5 + x_1 + x_3 + x_4 = 0 \\ x_1x_5 + x_2x_3 + x_3x_5 + x_5 + x_4 + x_2 + 1 = 0 \end{cases}$$

We perform Gaussian elimination on the system, and obtain,

$$\begin{cases} x_1x_2 + x_2x_4 + x_2x_5 + x_3x_5 + x_2 + x_3 + x_4 + x_5 = 0 \\ x_1x_3 + x_2x_4 + x_3 + 1 = 0 \\ x_1x_4 + x_2x_3 + x_3 + 1 = 0 \\ x_1x_5 + x_2x_3 + x_3x_5 + x_1 + x_3 + x_4 = 0 \\ x_2x_5 + x_3x_5 + x_4 + 1 = 0 \\ x_1 + x_2 + x_3 + x_5 + 1 = 0 \end{cases}$$

The linear equation we obtain is used for the substitution of variable $x_5 = x_1 + x_2 + x_3 + 1$. Then, we perform Gaussian elimination on the system again. We derive

$$\begin{cases} x_2x_4 + x_1 = 0 \\ x_1x_4 + x_2x_3 + x_3 + 1 = 0 \\ x_1x_2 + x_1 + x_3 + x_4 = 0 \\ x_1x_3 + x_1 + x_3 + 1 = 0 \\ x_4 + x_3 + x_1 = 0 \end{cases}$$

The new linear equation is used for the substitution of the variable $x_4 = x_3 + x_1$. After the substitution, we perform the Gaussian elimination again and obtain,

$$\begin{cases} x_2x_3 + x_1 = 0 \\ x_1x_3 + x_3 + 1 = 0 \\ x_1x_2 = 0 \\ x_1 = 0 \end{cases}$$

We derive a new linear equation $x_1 = 0$. Consequently, we perform substitution and Gaussian elimination, which yields,

$$\begin{cases} x_2 x_3 = 0 \\ x_3 + 1 = 0 \end{cases}$$

The new linear equation we obtain is $x_3 = 1$. After the substitution of this variable, we obtain $x_2 = 0$. Hence, we have gathered 5 linear equations in 5 variables as follows, which can be simply solved by

$$\begin{cases} x_1 + x_2 + x_3 + x_5 + 1 = 0 \\ x_1 + x_3 + x_4 = 0 \\ x_1 = 0 \\ x_3 + 1 = 0 \\ x_2 = 0 \end{cases}$$

leading to $x_1 = x_2 = x_5 = 0$ and $x_3 = x_4 = 1$.

B Multiple Samples Effect on ElimLin

A prominent question regarding ElimLin is that how we can extract more linear equations from the structure of the cipher. One approach is to use more samples. On one hand, having multiple instances increases the number of variables, since the state bits are totally distinct, on the other hand all the instances share the same key bits. The speed in which the number of equations increases is higher than which of the number of variables. Consequently, we expect that at one moment the system is solved. We have performed many experiments using ElimLin. In some cases, we would expect it to fail, since the number of linear equations at some stage dropped significantly, but those few equations could cause the system to collapse at the consequent iterations and the system is finally solved. We give an example to be more clear:

We attacked 8-round LBlock [50] block cipher with 32 LSB key bits fixed starting from 1 pair to 8 pairs (see Sec. 5.3 for details). As can be observed from Tables 4 to Table 11, the cipher is unbroken for 5 plaintext-ciphertext pairs, but then 6 pairs is enough to break the system. We use the following legend in those tables.

Legend
I : iteration number in ElimLin.
n : number of variables.
m_0 : number of initial equations.
AvS : the average number of monomials per equation (It represents the sparsity).
T : number of monomials.
n_L : number of linear equations.
n_c : cumulative number of linear equations.

For instance, in Table 9 we start with $n_0 = 8\,784$ variables and $m_0 = 27\,758$ equations. We then eliminate $n_L^1 = 7\,528$ variables at iteration 1. The variable elimination is repeated until the iteration 15, when we finish with 2 variables and $n_L^{15} = 2$ linear equations. As can be observed, at the last iteration the number of cumulative linear equations n_c is the same as the initial number of variables n_0. This implies that we only need to solve a linear system of equations in $8\,784$ variables and the system is solved. This is not the case for smaller number of samples. For instance, in Table 8, at the last iteration we finish with 499 variables and no linear equations. This implies that all 499 variables should be guessed.

Table 4. Attacking 8-round LBlock with 1 pair and 32 LSB key bits guessed.

I	n	m_0	AvS	T	n_L	n_c
1	2064	5174	3	4249	1768	1768
2	296	5174	7	5678	42	1810
3	254	5174	6	5035	16	1826
4	238	5174	7	4868	3	1829
5	235	5174	7	5178	0	1829

Table 5. Attacking 8-round LBlock with 2 pairs and 32 LSB key bits guessed.

I	n	m_0	AvS	T	n_L	n_c
1	3408	8822	3	7385	2920	2920
2	488	8822	8	10855	85	3005
3	403	8822	9	11545	48	3053
4	355	8822	11	11955	22	3027
5	333	8822	15	13779	8	3035
6	325	8822	16	13729	0	3035

Table 6. Attacking 8-round LBlock with 3 pairs and 32 LSB key bits guessed.

I	n	m_0	AvS	T	n_L	n_c
1	4752	13110	3	10521	4072	4072
2	680	13110	8	16032	128	4200
3	552	13110	9	17495	83	4283
4	469	13110	13	18190	40	4323
5	429	13110	17	19913	21	4344
6	408	13110	20	20547	5	4349
7	403	13110	22	20843	1	4350
8	402	13110	21	20725	1	4351
9	401	13110	21	20561	0	4351

Table 7. Attacking 8-round LBlock with 4 pairs and 32 LSB key bits guessed.

I	n	m_0	AvS	T	n_L	n_c
1	6096	17839	3	13657	5224	5224
2	872	17839	8	21209	171	5395
3	701	17839	10	24035	118	5511
4	583	17839	14	25396	66	5577
5	517	17839	20	27955	40	5617
6	477	17839	26	31106	21	5638
7	456	17839	31	31611	16	5654
8	440	17839	28	28934	1	5655
9	439	17839	28	28717	0	5655

Table 8. Attacking 8-round LBlock with 5 pairs and 32 LSB key bits guessed.

I	n	m_0	AvS	T	n_L	n_c
1	7440	22730	3	16793	6376	6376
2	1064	22730	8	26386	214	6590
3	850	22730	10	30368	151	6741
4	699	22730	15	33097	91	6832
5	608	22730	23	37005	55	6887
6	553	22730	32	39058	35	6922
7	518	22730	35	37629	16	6938
8	502	22730	34	35748	1	6939
9	501	22730	35	35709	1	6940
10	500	22730	34	35509	1	6941
11	499	22730	34	34649	0	6941

Table 9. Attacking 8-round LBlock with 6 pairs and 32 LSB key bits guessed.

I	n	m_0	AvS	T	n_L	n_c
1	8784	27758	3	19929	7528	7528
2	1256	27758	7	31563	257	7785
3	999	27758	10	36607	189	7974
4	810	27758	17	41351	123	8097
5	687	27758	26	48066	83	8180
6	604	27758	34	46540	41	8221
7	563	27758	36	42910	15	8236
8	548	27758	37	41469	8	8244
9	540	27758	32	39312	24	8268
10	516	27758	16	29409	126	8394
11	390	27758	19	23370	108	8502
12	282	27758	20	14889	87	8589
13	195	27758	15	9157	122	8711
14	73	27758	4	1454	71	8782
15	2	27758	0	3	2	8784

Table 10. Attacking 8-round LBlock with 7 pairs and 32 LSB key bits guessed.

I	n	m_0	AvS	T	n_L	n_c
1	10128	32815	3	23065	8680	8680
2	1448	32815	7	36740	300	8980
3	1148	32815	10	42889	228	9208
4	920	32815	17	48974	157	9365
5	763	32815	30	58471	111	9476
6	652	32815	40	55476	47	9523
7	605	32815	42	51967	20	9543
8	585	32815	37	47625	25	9568
9	560	32815	19	36163	141	9709
10	419	32815	21	27254	126	9835
11	293	32815	20	16116	145	9980
12	148	32815	8	4960	142	10122
13	6	32815	0	8	6	10128

Table 11. Attacking 8-round LBlock with 8 pairs and 32 LSB key bits guessed.

I	n	m_0	AvS	T	n_L	n_c
1	11472	37945	3	26201	9832	9832
2	1640	37945	7	41917	343	10175
3	1297	37945	9	47974	268	10443
4	1029	37945	17	55084	186	10629
5	843	37945	30	65625	129	10758
6	714	37945	39	63385	57	10815
7	657	37945	41	57671	22	10837
8	635	37945	34	50898	21	10858
9	614	37945	20	40883	161	11019
10	453	37945	23	30905	144	11163
11	309	37945	22	19850	160	11323
12	149	37945	8	5108	145	11468
13	4	37945	0	6	4	11472