ElimLin Algorithm Revisited

Nicolas Courtois, Pouyan Sepehrdad, Petr Sušil and Serge Vaudenay

University College London, UK

EPFL, Lausanne, Switzerland

Algebraic Cryptanalysis

- Solving multivariate polynomial representation of a given cipher.
- Initially recognized by Shannon back in 1949.
- Been used to attack various stream ciphers and less frequently block ciphers.
- Requires small number of plaintext-ciphertext pairs.
- The problem of solving multivariate polynomial systems of equations is NP hard and is referred to as MQ problem.

Distinct approaches to solve the MQ problem

- Random systems: same number of equations as unknowns
  - No algorithm better than exhaustive search.
- Overdefined systems:
  - Gröbner basis (F4 and F5), XL and MutantXL.
- Sparse systems:
  - XSL, SAT solvers, Raddum-Semaev algorithm and ElimLin.

Distinct approaches to solve the MQ problem

- Random systems: same number of equations as unknowns
  - No algorithm better than exhaustive search.
- Overdefined systems:
  - Gröbner basis (F4 and F5), XL and MutantXL.
- Sparse systems:
  - XSL, SAT solvers, Raddum-Semaev algorithm and ElimLin.

Goals: What can we learn from the theory behind ElimLin?
Is it better than the other heuristic techniques or is it worse?
Do we always need sophisticated techniques such as Gröbner basis?

ElimLin Algorithm

- Stands for **Elim**inate **Lin**ear
- Also known as "inter-reduction" in all major algebra systems.
- Devised as a single tool by Courtois to attack DES (2006).
- An algorithm for solving multivariate polynomial systems.
- The degree of the system is maintained during all the steps.
- Finds some **(NOT ALL)** hidden linear equations in the system.
- An extremely simple algorithm, but hard to analyze.
- Almost nothing has been done regarding the theory behind it.
- Might be sufficient for large number of samples.

ElimLin (a very simple algorithm)

Repeat

- *Gaussian Elimination:* All the linear equations in the linear span of initial equations are found.
- *Substitution:* Variables are iteratively eliminated in the whole system based on linear equations until there is no linear equation left.

Until no new linear equation is obtained in the linear span of the system.

Why does ElimLin generate new linear equations?

- Linearization is done first.
- Gaussian elimination does not capture the relation between the monomials.
- After the substitution, new monomials may appear.
- Depending on the substitution, the system we obtain is different.
- Intuitively, depending on the substitution, the result of ElimLin might be different.
- Gröbner basis algorithms: ordering as a prominent factor

Do distinct substitutions and Gaussian eliminations or any bijective affine variable change modify the ElimLin result?

- Intuitively, depending on the substitution, the result of ElimLin might be different.
- Gröbner basis algorithms: Ordering as a prominent factor.

Do distinct substitutions and Gaussian eliminations or any bijective affine variable change modify the ElimLin result?

- Intuitively, depending on the substitution, the result of ElimLin might be different.
- Gröbner basis algorithms: Ordering as a prominent factor.

NO!

See the paper for the proof.

Characterizing the ElimLin Output

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: $\quad \bar{\mathcal{S}}_L \leftarrow \mathrm{Span}\left(\mathcal{S}^0 \cup (R_{d-1} \times \bar{\mathcal{S}}_L)\right) \cap R_1$
6: **until** $\bar{\mathcal{S}}_L$ unchanged
7: Output $\mathcal{S}_L$: a basis of $\bar{\mathcal{S}}_L$.

where

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

Does ElimLin generate all the hidden linear equations? NO!

An example, demonstrating such an evidence:

- $\ell(x)g(x) + 1 = 0$ over GF(2), where $\ell(x)$ is a polynomial of degree one and $g(x)$ is a polynomial of degree at most $d - 1$.
- ElimLin on this single equation trivially fails.
- Multiply both sides of the equation by $\ell(x) + 1$.
- We obtain: $\ell(x) = 1$.

Does ElimLin generate all the hidden linear equations? NO!

An example, demonstrating such an evidence:

- $\ell(x)g(x) + 1 = 0$ over GF(2), where $\ell(x)$ is a polynomial of degree one and $g(x)$ is a polynomial of degree at most $d - 1$.
- ElimLin on this single equation trivially fails.
- Multiply both sides of the equation by $\ell(x) + 1$.
- We obtain: $\ell(x) = 1$.

ElimLin can be extended!

Does ElimLin generate all the hidden linear equations? NO!

An example, demonstrating such an evidence:

- $\ell(x)g(x) + 1 = 0$ over GF(2), where $\ell(x)$ is a polynomial of degree one and $g(x)$ is a polynomial of degree at most $d - 1$.
- ElimLin on this single equation trivially fails.
- Multiply both sides of the equation by $\ell(x) + 1$.
- We obtain: $\ell(x) = 1$.

ElimLin can be extended! (The extension is degree 2-bounded Gröbner basis)

How can we extract more linear equations from the structure of the cipher?

Using more samples: **NOT** independent!

- Multiple instances of the same system with different input/output pairs. Does it help?
- The instances only share the key variables.
- With more samples: the number of equations increases faster than the number of variables.
- We expect the system to collapse when we have enough number of samples.

How can we extract more linear equations from the structure of the cipher?

Using more samples: **NOT** independent!

- Multiple instances of the same system with different input/output pairs. Does it help?
- The instances only share the key variables.
- With more samples: the number of equations increases faster than the number of variables.
- We expect the system to collapse when we have enough number of samples.

How many samples? (still an open problem)

Some Lightweight Block Ciphers

Later, we attack a few lightweight block ciphers:

- LBlock: a new lightweight Feistel-based block cipher, proposed at ACNS 2011, operates on 64-bit blocks, uses a key of 80 bits and has 32 rounds.
- MIBS: a lightweight Feistel-based block cipher, proposed at CANS 2009, operates on 64-bit blocks, uses a key of 64 or 80 bits and has 32 rounds.
- CTC2: a toy SPN-based block cipher devised by Courtois to evaluate the security of ciphers with respect to algebraic attacks, variable block and key sizes.

Table: 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: 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

Is ElimLin a powerful algorithm?

- ElimLin is a polynomial time algorithm $O(n_0^5)$ in the initial number of variables.
- Can we show that with polynomial number of samples ElimLin succeeds?
  - Both yes/no answers are of great interest.
  - Yes: already a breakthrough in cryptography.
  - No: for instance, AES can never be broken by algebraic attacks at degree 2.

Is ElimLin a powerful algorithm?

- ElimLin is a polynomial time algorithm $O(n_0^5)$ in the initial number of variables.
- Can we show that with polynomial number of samples ElimLin succeeds?
  - Both yes/no answers are of great interest.
  - Yes: already a breakthrough in cryptography.
  - No: for instance, AES can never be broken by algebraic attacks at degree 2.

Lack of theory, need a very efficient implementation of ElimLin and in general, the Gröbner basis algorithms.

Gröbner Basis Algorithms

- Gröbner basis and SAT solving techniques: the most successful methods to solve polynomial systems.
- Both have their own restrictions.
- Gröbner basis approach often needs a large amount of memory.
- Faster than other methods for overdefined dense systems and when the system is over GF(q), where $q > 2$.
- Sometimes faster for small systems over GF(2).

Gröbner Basis Algorithms

- Gröbner basis and SAT solving techniques: the most successful methods to solve polynomial systems.
- Both have their own restrictions.
- Gröbner basis approach often needs a large amount of memory.
- Faster than other methods for overdefined dense systems and when the system is over GF(q), where $q > 2$.
- Sometimes faster for small systems over GF(2).

Do we always need sophisticated methods such as Gröbner basis algorithms?

F4 under PolyBoRi Framework

- PolyBoRi: the most efficient implementation of F4 algorithm (free compared to Magma).
- A C++ library with a Python interface.
- Uses zero-suppressed binary decision diagrams (ZDDs) to store polynomials efficiently in memory.
- It also makes the Gröbner basis computation faster.
- We used polybori-0.8.0 for our experiments.

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

N_r	$\#key$	g	Running Time (in hours)	Data	Attack notes	CPU	RAM
8	80	32	0.252	6 KP	ElimLin	2.8 Ghz	4 GB
8	80	32	crashed!	6 KP	PolyBoRi	2.8 Ghz	4 GB

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

Table: Algebraic attack complexities on reduced-round MIBS using ElimLin and PolyBoRi.

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 : the number of rounds
g: the number of guessed LSB of the key
KP: known plaintext

Going to the higher degrees

- Sometimes, it is enough to stay at degree 2.
- ElimLin solves the system at degree 2.
- Gröbner basis algorithms often break the systems at higher degrees.
- ElimLin is a part of Gröbner basis computations for a specific ordering.
- Since memory is limited, maybe we should not go to higher degrees.
- We used an efficient implementation of the ElimLin algorithm (might not be efficient enough).

- We shed some light on to how ElimLin works.
- We presented a unique characterization of the set of linear equations in ElimLin in terms of a fixed point.
- Variable ordering or any affine bijective variable change does not modify the result of ElimLin.
- Much more research needs to be done regarding ElimLin:
  - Have a better understanding of how the algorithm works.
  - How the number of linear equations evolves in each iteration?
  - Is polynomial number of samples enough for ElimLin to work?
  - Does ElimLin always succeed with infinite number of samples?
  - At what degree, a system is solvable by ElimLin?
  - Finding a more efficient implementation of the algorithm.

Questions?