When POlynomial System SOlving became a threat for symmetric primitives

Léo Perrin[1]

[1]Inria, France

ZKCS 2026, February 2026, Vienna

informatics mathematics

Algebraic attack?

What do we mean by that?

Algebraic attack? (1/3)

Algebraic attack?

GEOMETRIC ASPECTS OF THE ABELIAN MODULAR FUNCTIONS OF GENUS FOUR (I)

BY ARTHUR B. COBLE[1]

DEPARTMENT OF MATHEMATICS, UNIVERSITY OF ILLINOIS

Communicated by E. H. Moore, June 21, 1921

1. *Introduction.*—The plane curve of genus 4 has a canonical series g_3^6 and is mapped from the plane by the canonical adjoints into the *normal* curve of genus 4, a space sextic which is the complete intersection of a quadric and a cubic surface. If we denote a point of this quadric by the parameters t, τ of the cross generators through it the equation of this sextic is $F = (a\tau)^3 (\alpha t)^3 = 0$. For geometric purposes we may define a modular function to be any rational or irrational invariant of the form F, bi-cubic in the digredient binary variables τ, t; for transcendental purposes it is desirable to restrict this definition by requiring further that this invariant, regarded as a function of the normalized periods ω_{ij} of the abelian integrals attached to the curve, be uniform.

There seems to be an unusually rich variety of geometric entities which center about this normal curve. Some of these have received independent investigation. It is the purpose of this series of abstracts to indicate a number of new relations among these various entities and to connect each with the normal sextic F. The methods employed are in the main geometric. Direct algebraic attack on problems which contain nine irremovable constants, or moduli, is difficult. However much information is gained by a free use of algebraic forms containing sets of variables drawn from different domains. Both finite and infinite discontinuous groups are utilized at various times.

Coble, A. B. (1921). *Geometric Aspects of the Abelian Modular Functions of Genus Four (I)*. Proceedings of the National Academy of Sciences.

Algebraic attack? (2/3)

Algebraic attack?

Algebraic Attacks on Stream Ciphers with Linear Feedback

Nicolas T. Courtois[1] and Willi Meier[2]

[1] Cryptography Research, Schlumberger Smart Cards, 36-38 rue de la Princesse, BP 45, F-78430 Louveciennes Cedex, France, courtois@minrank.org
[2] FH Aargau, CH-5210 Windisch, Switzerland, meierw@fh-argau.ch

Abstract. A classical construction of stream ciphers is to combine several LFSRs and a highly non-linear Boolean function f. Their security is usually analysed in terms of correlation attacks, that can be seen as solving a system of multivariate linear equations, true with some probability. At ICISC'02 this approach is extended to systems of higher-degree multivariate equations, and gives an attack in 2^{92} for Toyocrypt, a Cryptrec submission. In this attack the key is found by solving an overdefined system of algebraic equations. In this paper we show how to substantially lower the degree of these equations by multiplying them by well-chosen multivariate polynomials. Thus we are able to break Toyocrypt in 2^{49} CPU clocks, with only 20 Kbytes of keystream, the fastest attack proposed so far. We also successfully attack the Nessie submission LILI-128, within 2^{57} CPU clocks (not the fastest attack known). In general, we show that if the Boolean function uses only a small subset (e.g. 10) of state/LFSR bits, the cipher can be broken, whatever is the Boolean function used (worst case). Our new general algebraic attack breaks stream ciphers satisfying all the previously known design criteria in at most the square root of the complexity of the previously known generic attack.

Courtois, N. T., & Meier, W. (2003, May). *Algebraic attacks on stream ciphers with linear feedback*. EUROCRYPT'03.

Algebraic attack? (3/3)

Algebraic attack?

2. Unbalanced Oil and Vinegar Scheme

The most interesting type of one-way function used in multivariate cryptography is based on the evaluation of a set of algebraic polynomials $\mathbf{p} = (p_1(x_1, \ldots, x_n), \ldots, p_m(x_1, \ldots, x_n)) \in \mathbb{K}[x_1, \ldots, x_n]^m$, namely :

$$\mathbf{m} = (m_1, \ldots, m_n) \in \mathbb{K}^n \longmapsto \mathbf{p}(\mathbf{m}) = (p_1(\mathbf{m}), \ldots, p_m(\mathbf{m})) \in \mathbb{K}^m.$$

The mathematical hard problem underlying this one-way function is :

Polynomial System Solving (PoSSo)

INSTANCE : polynomials $p_1(x_1, \ldots, x_n), \ldots, p_m(x_1, \ldots, x_n)$ of $\mathbb{K}[x_1, \ldots, x_n]$.

QUESTION : Does there exists $(z_1, \ldots, z_n) \in \mathbb{K}^n$ s. t. :

$$p_1(z_1, \ldots, z_n) = 0, \ldots, p_m(z_1, \ldots, z_n) = 0.$$

Bettale, L., Faugère, J. C., & Perret, L. (2012, July). *Solving polynomial systems over finite fields: improved analysis of the hybrid approach*. ISSAC.

$$\begin{aligned}\text{Algebraic attack} &= \text{Writing equations} + \text{external solver}\\ &= \text{POlynomial System} + \text{Solving}\end{aligned}$$

$$
\begin{aligned}
\text{Algebraic attack} &= \text{Writing equations} + \text{external solver} \\
&= \text{POlynomial System} + \text{Solving}
\end{aligned}
$$

From a modern symmetric cryptography perspective

Should we care?

$$\text{Algebraic attack} = \text{Writing equations} + \text{external solver}$$
$$= \text{POlynomial System} + \text{Solving}$$

From a modern symmetric cryptography perspective

Should we care? ... clearly **yes**

How should we care?

$$\text{Algebraic attack} = \text{Writing equations} + \text{external solver}$$
$$= \text{POlynomial System} + \text{Solving}$$

From a modern symmetric cryptography perspective

Should we care? ... clearly **yes**

How should we care? **... by systemizing the knowledge we have**

$$\text{Algebraic attack} = \text{Writing equations} + \text{external solver}$$
$$= \text{POlynomial System} + \text{Solving}$$

From a modern symmetric cryptography perspective

Should we care? ... clearly **yes**

How should we care? **... by systemizing the knowledge we have**

Conclusion

We need to greatly simplify the study of:

1. what PoSSo techniques are, and how efficient they are
2. the polynomial systems arising from symmetric cryptanalysis

Outline

Plan of this Section

Plan of this Section

1 Intro: What do we call an algebraic attack?

2 What is a PoSSo-based Attack?
- Attacking Elisabeth
- Outline of an attack
- Security Arguments

3 The More Complicated Case of "Real" PoSSo

4 Where to go from here?

The particular case of Elizabeth (1/2)

Elisabeth

Implements a “filter permutator”

- Key register is never modified
- Optimized for TFHE
- Low multiplicative depth: F is of low degree

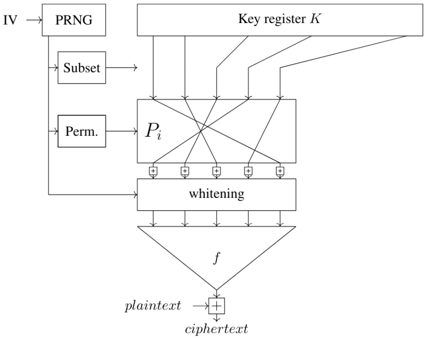


Fig. 1: The group filter permutator design

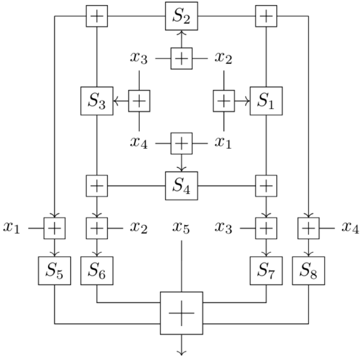


Fig. 2: Elisabeth-4 ’s 5 to 1 inner function.

The particular case of Elisabeth (2/2)

Basic Linearization

$$
\begin{aligned}
s_i &= F(x_0, \dots x_\ell) \\
&= \sum_{u \in \mathbb{F}_2^n} \alpha_u \prod_j x_j^{u_j} = \sum_{u \in \mathbb{F}_2^n} \alpha_u P_u
\end{aligned}
$$

α_u is known, both x_j and P_u must be recovered.

The particular case of Elisabeth (2/2)

Basic Linearization

$$
\begin{aligned}
s_i &= F(x_0, \ldots x_\ell) \\
&= \sum_{u \in \mathbb{F}_2^n} \alpha_u \prod_j x_j^{u_j} = \sum_{u \in \mathbb{F}_2^n} \alpha_u P_u
\end{aligned}
$$

α_u is known, both x_j and P_u must be recovered.
If the number of equations is high enough, this can be solved using basic linear algebra!

The particular case of Elisabeth (2/2)

Basic Linearization

$$s_i = F(x_0, \ldots x_\ell)$$
$$= \sum_{u \in \mathbb{F}_2^n} \alpha_u \prod_j x_j^{u_j} = \sum_{u \in \mathbb{F}_2^n} \alpha_u P_u$$

α_u is known, both x_j and P_u must be recovered. If the number of equations is high enough, this can be solved using basic linear algebra!

Improvement

1. Careful analysis of the ANF means there are fewer unknowns
2. Better algorithm than Gaussian elimination

The particular case of Elisabeth (2/2)

Basic Linearization

$$s_i = F(x_0, \ldots x_\ell) = \sum_{u \in \mathbb{F}_2^n} \alpha_u \prod_j x_j^{u_j} = \sum_{u \in \mathbb{F}_2^n} \alpha_u P_u$$

α_u is known, both x_j and P_u must be recovered. If the number of equations is high enough, this can be solved using basic linear algebra!

Improvement

1. Careful analysis of the ANF means there are fewer unknowns
2. Better algorithm than Gaussian elimination

Model	Time (operations)	Data (nibbles)	Memory (bits)
Known IV	2^{124}	2^{43}	2^{87}
Known IV	2^{116}	2^{41}	2^{81}
Known IV	2^{94}	2^{41}	2^{57}
Known IV	2^{88}	2^{87}	2^{54}
Chosen IV	2^{88}	2^{37}	2^{54}

Plan of this Section

The Steps of an "Algebraic Attack"

Write a system of equations.

The Steps of an “Algebraic Attack”

Write a system of equations.

Solve system. If the system is completely linear, trivial.
If not, maybe linearize?

The Steps of an “Algebraic Attack”

Write a system of equations.

Solve system. If the system is completely linear, trivial.
If not, maybe linearize? Or something else!

The Steps of an "Algebraic Attack"

Write a system of equations.

Solve system. If the system is completely linear, trivial.
If not, maybe linearize? Or something else!

Deduce attack from result. (e.g. recover secret key from variable assignment)

The Steps of an "Algebraic Attack"

Initial Cryptanalysis. Deduce a system of equations as simple as possible from the intended attack.

Write a system of equations.

Solve system. If the system is completely linear, trivial.
If not, maybe linearize? Or something else!

Deduce attack from result. (e.g. recover secret key from variable assignment)

Plan of this Section

A failed argument

would be more costly in this context. On a filtered LFSR the security estimation is $\mathcal{O}(D\log^2(D) + ED\log(D) + E^\omega)$ where $D = \sum_{i=1}^{\deg(h)} \binom{N}{i}$ and $E = \sum_{i=1}^{\deg(g)} \binom{N}{i}$. This estimation will be used as an indicator rather than a sharp limit, considering that the complexity of the best attack of the algebraic kind would lie between this (too low) bound and the (too high) one given by the algebraic attack of Courtois-Meier.

A failed argument

would be more costly in this context. On a filtered LFSR the security estimation is $\mathcal{O}(D\log^2(D) + ED\log(D) + E^\omega)$ where $D = \sum_{i=1}^{\deg(h)} \binom{N}{i}$ and $E = \sum_{i=1}^{\deg(g)} \binom{N}{i}$. This estimation will be used as an indicator rather than a sharp limit, considering that the complexity of the best attack of the algebraic kind would lie between this (too low) bound and the (too high) one given by the algebraic attack of Courtois-Meier.

What went wrong?

1. Wrong assumptions about the relevant metric: degree vs. number of possible monomials
2. Attack complexity was further lowered using sophisticated (but still off-the-shelf) algorithms (block Wiedemann)

Plan of this Section

Definition

Polynomial System Solving

$$\begin{cases} p_1(x_1, \ldots, x_N) = 0 \\ \vdots \\ p_{k-1}(x_1, \ldots, x_N) = 0 \\ p_k(x_1, \ldots, x_N) = 0 \end{cases}$$

Definition

Polynomial System Solving

$$\begin{cases} p_1(x_1, \ldots, x_N) = 0 \\ \vdots \\ p_{k-1}(x_1, \ldots, x_N) = 0 \\ p_k(x_1, \ldots, x_N) = 0 \end{cases} \longrightarrow$$

PoSSo
magic!

Definition

Polynomial System Solving

$$\begin{cases} p_1(x_1,\ldots,x_N) = 0 \\ \vdots \\ p_{k-1}(x_1,\ldots,x_N) = 0 \\ p_k(x_1,\ldots,x_N) = 0 \end{cases} \quad \xrightarrow[\text{PoSSo magic!}]{} \quad \begin{cases} x_1 = c_1 \\ \vdots \\ x_{N-1} = c_{N-1} \\ x_N = c_N \end{cases}$$

Here, we suppose that there is a finite, small number of solutions (e.g. not a full vector space).

Plan of this Section

Symmetric Cryptanalysis

CICO-1

Symmetric Cryptanalysis

CICO-1

and also...

CICO-k, limited birthday, actual preimages

Symmetric Cryptanalysis

CICO-1

and also...

CICO-k, limited birthday, actual preimages

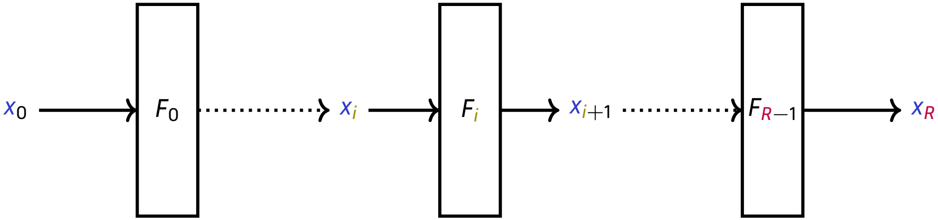
and also...

block/stream cipher key recovery

Even more cases!

Variety in solving techniques is matched by an even greater variety in attack type

CICO-1 for a Permutation (Naïve case)



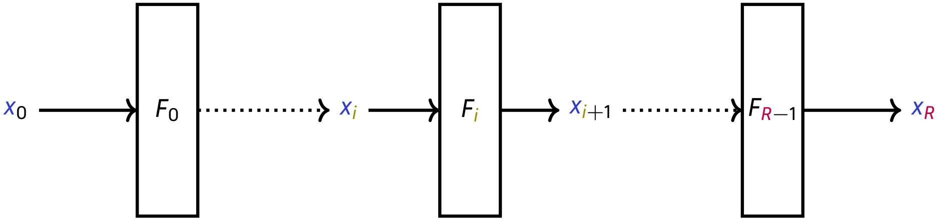
Naïve Encoding

Sub-variables x_i for each round;

$x_i = (x_{i,0}, \ldots, x_{i,\ell-1})$

$$\begin{cases} x_0 & = (?, \ldots, ?, 0) \\ \ldots & \ldots \\ x_{i+1} & = F_i(x_i) \\ \ldots & \ldots \\ x_R & = (?, .., ?, 0) \end{cases}$$

CICO-1 for a Permutation (Naïve case)



Naïve Encoding

Sub-variables x_i for each round;

$$x_i = (x_{i,0}, \ldots, x_{i,\ell-1})$$

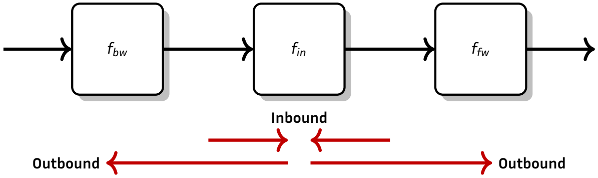
$$\begin{cases} x_0 & = (?, \ldots, ?, 0) \\ \ldots & \ldots \\ x_{i+1} & = F_i(x_i) \\ \ldots & \ldots \\ x_R & = (?, .., ?, 0) \end{cases}$$

$$\begin{cases} x_0 & = (?, \ldots, ?, 0) \\ \ldots & \ldots \\ F_i{}'(x_i,\ x_{i+1}) & = 0 \\ \ldots & \ldots \\ x_R & = (?, \ldots, ?, 0) \end{cases}$$

Limited Birthday Distinguisher (via Rebound Attack)

Goal: find (x, x') such that

$$x + x' \in V \qquad f(x) + f(x') \in W$$



Rebound Attack

1. Generate a lot of pair inner values y, y'
2. Hope that they propagate to good $x, x', f(x), f(x')$

Step 1. (inbound) can be done by solving a system!

Plan of this Section

Generic System Solving: Steps and Tools

$$\begin{cases} p_1(x_1, \ldots, x_N) = 0 \\ \vdots \\ p_{k-1}(x_1, \ldots, x_N) = 0 \\ p_k(x_1, \ldots, x_N) = 0 \end{cases}$$

1. Define system

The Tools Used

SAGE Open source, mostly reliable...

Generic System Solving: Steps and Tools

$$\begin{cases} p_1(x_1, \ldots, x_N) = 0 \\ \vdots \\ p_{k-1}(x_1, \ldots, x_N) = 0 \\ p_k(x_1, \ldots, x_N) = 0 \end{cases}$$

1. Define system

The Tools Used

SAGE Open source, mostly reliable... **but** sloooow

Generic System Solving: Steps and Tools

$$\begin{cases} p_1(x_1, \ldots, x_N) = 0 \\ \vdots \\ p_{k-1}(x_1, \ldots, x_N) = 0 \\ p_k(x_1, \ldots, x_N) = 0 \end{cases} \qquad \begin{cases} g_1(x_1, \ldots, x_N) = 0 \\ \vdots \\ g_{\kappa-1}(x_1, \ldots, x_N) = 0 \\ g_\kappa(x_1, \ldots, x_N) = 0 \end{cases}$$

1. Define system
2. Find a GB (F4/F5)

The Tools Used

SAGE Open source, mostly reliable... **but** sloooow

F4/F5 What makes it possible...

Generic System Solving: Steps and Tools

$$\begin{cases} p_1(x_1, \ldots, x_N) = 0 \\ \vdots \\ p_{k-1}(x_1, \ldots, x_N) = 0 \\ p_k(x_1, \ldots, x_N) = 0 \end{cases} \qquad \begin{cases} g_1(x_1, \ldots, x_N) = 0 \\ \vdots \\ g_{\kappa-1}(x_1, \ldots, x_N) = 0 \\ g_\kappa(x_1, \ldots, x_N) = 0 \end{cases}$$

1. Define system
2. Find a GB (F4/F5)

The Tools Used

SAGE Open source, mostly reliable... **but** slooooow

F4/F5 What makes it possible...**but** Open source implementations hard to find, big difference in efficiency between public/proprietary implementations.

Generic System Solving: Steps and Tools

$$\begin{cases} p_1(x_1, \ldots, x_N) = 0 \\ \vdots \\ p_{k-1}(x_1, \ldots, x_N) = 0 \\ p_k(x_1, \ldots, x_N) = 0 \end{cases} \qquad \begin{cases} g_1(x_1, \ldots, x_N) = 0 \\ \vdots \\ g_{\kappa-1}(x_1, \ldots, x_N) = 0 \\ g_\kappa(x_1, \ldots, x_N) = 0 \end{cases}$$

1. Define system
2. Find a GB (F4/F5)

The Tools Used

SAGE Open source, mostly reliable... **but** slooooow

F4/F5 What makes it possible...**but** Open source implementations hard to find, big difference in efficiency between public/proprietary implementations.

but things are getting better!

Generic System Solving: Steps and Tools

$$\begin{cases} p_1(x_1,\ldots,x_N)=0 \\ \vdots \\ p_{k-1}(x_1,\ldots,x_N)=0 \\ p_k(x_1,\ldots,x_N)=0 \end{cases} \qquad \begin{cases} g_1(x_1,\ldots,x_N)=0 \\ \vdots \\ g_{\kappa-1}(x_1,\ldots,x_N)=0 \\ g_\kappa(x_1,\ldots,x_N)=0 \end{cases} \qquad \begin{cases} g_1^*(x_1,\ldots,x_N)=0 \\ \vdots \\ g_{N-1}^*(x_{N-1},x_N)=0 \\ g_N^*(x_N)=0 \end{cases}$$

1. Define system
2. Find a GB (F4/F5)
3. Change order to **lex**

The Tools Used

SAGE Open source, mostly reliable... **but** slooooow

F4/F5 What makes it possible...**but** Open source implementations hard to find, big difference in efficiency between public/proprietary implementations.

but things are getting better!

FGLM No "practical" issues...

Generic System Solving: Steps and Tools

$$\begin{cases} p_1(x_1, \ldots, x_N) = 0 \\ \vdots \\ p_{k-1}(x_1, \ldots, x_N) = 0 \\ p_k(x_1, \ldots, x_N) = 0 \end{cases} \qquad \begin{cases} g_1(x_1, \ldots, x_N) = 0 \\ \vdots \\ g_{\kappa-1}(x_1, \ldots, x_N) = 0 \\ g_\kappa(x_1, \ldots, x_N) = 0 \end{cases} \qquad \begin{cases} g_1^*(x_1, \ldots, x_N) = 0 \\ \vdots \\ g_{N-1}^*(x_{N-1}, x_N) = 0 \\ g_N^*(x_N) = 0 \end{cases}$$

1. Define system
2. Find a GB (F4/F5)
3. Change order to **lex**

The Tools Used

SAGE Open source, mostly reliable... **but** sloooow

F4/F5 What makes it possible...**but** Open source implementations hard to find, big difference in efficiency between public/proprietary implementations.

but things are getting better!

FGLM No “practical” issues... **but** lots of variants with very different complexities

Generic System Solving: Steps and Tools

$$\begin{cases} p_1(x_1,\ldots,x_N) = 0 \\ \vdots \\ p_{k-1}(x_1,\ldots,x_N) = 0 \\ p_k(x_1,\ldots,x_N) = 0 \end{cases} \quad \begin{cases} g_1(x_1,\ldots,x_N) = 0 \\ \vdots \\ g_{\kappa-1}(x_1,\ldots,x_N) = 0 \\ g_\kappa(x_1,\ldots,x_N) = 0 \end{cases} \quad \begin{cases} g_1^*(x_1,\ldots,x_N) = 0 \\ \vdots \\ g_{N-1}^*(x_{N-1},x_N) = 0 \\ g_N^*(x_N) = 0 \end{cases} \quad g_N^*(x_N) = 0 \rightarrow x_N$$

1. Define system
2. Find a GB (F4/F5)
3. Change order to **lex**
4. Univariate root

The Tools Used

SAGE: Open source, mostly reliable... **but** sloooow

F4/F5: What makes it possible...**but** Open source implementations hard to find, big difference in efficiency between public/proprietary implementations.

but things are getting better!

FGLM: No "practical" issues... **but** lots of variants with very different complexities

Berlekamp-Rabin: Easy to re-implement

A Better Solving Strategy (Sometimes): the Freelunch

$$\begin{cases} p_1(x_1, \ldots, x_N) = 0 \\ \ldots \\ p_{N-1}(x_1, \ldots, x_N) = 0 \\ p_N(x_1, \ldots, x_N) = 0 \end{cases}$$

1. Encoding

$$\begin{cases} g_0(x_1, \ldots, x_N) = 0 \\ \ldots \\ g_{N-1}(x_1, \ldots, x_N) = 0 \\ g_N(x_1, \ldots, x_N) = 0 \end{cases}$$

2. Find Gröbner basis

$$\begin{cases} g_1^*(x_1, \ldots, x_N) = 0 \\ \ldots \\ g_{N-1}^*(x_{N-1}, x_N) = 0 \\ g_N^*(x_N) = 0 \end{cases}$$

3. Change order

find x_0

...

deduce all x_i:s

4. Final resolution

A Better Solving Strategy (Sometimes): the Freelunch

$$\begin{cases} p_1(x_1, \ldots, x_N) = 0 \\ \ldots \\ p_{N-1}(x_1, \ldots, x_N) = 0 \\ p_N(x_1, \ldots, x_N) = 0 \end{cases}$$

$$\begin{cases} g_0(x_1, \ldots, x_N) = 0 \\ \ldots \\ g_{N-1}(x_1, \ldots, x_N) = 0 \\ g_N(x_1, \ldots, x_N) = 0 \end{cases}$$

$$\begin{cases} g_1^*(x_1, \ldots, x_N) = 0 \\ \ldots \\ g_{N-1}^*(x_{N-1}, x_N) = 0 \\ g_N^*(x_N) = 0 \end{cases}$$

find x_0

...

deduce all x_i:s

1. Encoding
2. Find Gröbner basis
3. Change order
4. Final resolution

A Better Solving Strategy (Sometimes): the Freelunch

$$\begin{cases} p_1(x_1, \ldots, x_N) = 0 \\ \ldots \\ p_{N-1}(x_1, \ldots, x_N) = 0 \\ p_N(x_1, \ldots, x_N) = 0 \end{cases}$$

$$\begin{cases} g_0(x_1, \ldots, x_N) = 0 \\ \ldots \\ g_{N-1}(x_1, \ldots, x_N) = 0 \\ g_N(x_1, \ldots, x_N) = 0 \end{cases}$$

→ Compute M_{x_0}

find x_0

...

deduce all x_i:s

1. Encoding
2. Find Gröbner basis
2. Get a matrix
4. Final resolution

A Better Solving Strategy (Sometimes): the Freelunch

$$\begin{cases} p_1(x_1, \ldots, x_N) = 0 \\ \ldots \\ p_{N-1}(x_1, \ldots, x_N) = 0 \\ p_N(x_1, \ldots, x_N) = 0 \end{cases}$$

$$\begin{cases} g_0(x_1, \ldots, x_N) = 0 \\ \ldots \\ g_{N-1}(x_1, \ldots, x_N) = 0 \\ g_N(x_1, \ldots, x_N) = 0 \end{cases}$$

→

Compute M_{x_0}

$\det(x_0 I - M_{x_0}) = 0$

1. Encoding
2. Find Gröbner basis
2. Get a matrix
3. Partial resolution

A Better Solving Strategy (Sometimes): the Freelunch

$$\begin{cases} p_1(x_1, \dots, x_N) = 0 \\ \dots \\ p_{N-1}(x_1, \dots, x_N) = 0 \\ p_N(x_1, \dots, x_N) = 0 \end{cases}$$

$$\begin{cases} g_0(x_1, \dots, x_N) = 0 \\ \dots \\ g_{N-1}(x_1, \dots, x_N) = 0 \\ g_N(x_1, \dots, x_N) = 0 \end{cases}$$

Compute M_{x_0}

$\det(x_0 I - M_{x_0}) = 0$

1. Encoding
2. Find Gröbner basis
2. Get a matrix
3. Partial resolution

What do we need?

1. Free Gröbner basis
2. Compute M_{x_0}
3. Solve $\det(x_0 I - M_{x_0}) = 0$

A Better Solving Strategy (Sometimes): the Freelunch

$$\begin{cases} p_1(x_1, \ldots, x_N) = 0 \\ \ldots \\ p_{N-1}(x_1, \ldots, x_N) = 0 \\ p_N(x_1, \ldots, x_N) = 0 \end{cases} \quad \begin{cases} g_0(x_1, \ldots, x_N) = 0 \\ \ldots \\ g_{N-1}(x_1, \ldots, x_N) = 0 \\ g_N(x_1, \ldots, x_N) = 0 \end{cases}$$

Compute M_{x_0}

$\det(x_0 I - M_{x_0}) = 0$

1. Encoding
2. Find Gröbner basis
2. Get a matrix
3. Partial resolution

What do we need?

1. Free Gröbner basis: the FreeLunch
2. Compute M_{x_0}: regular Gröbner basis arithmetic
3. Solve $\det(x_0 I - M_{x_0}) = 0$: dedicated algorithm

The Resultant-based Approach

$$\begin{cases} f(x_1, x_2) = 0 \\ g(x_1, x_2) = 0 \end{cases}$$

1. Define system

The Resultant-based Approach

$$Syl(f,g) = \underbrace{\begin{bmatrix} a_\gamma & \cdots & a_1 & a_0 & & 0 \\ & \ddots & & \ddots & \ddots & \\ 0 & & a_\gamma & \cdots & a_1 & a_0 \\ b_\delta & b_{\delta-1} & \cdots & b_0 & & 0 \\ & \ddots & \ddots & & \ddots & \\ 0 & & b_\delta & b_{\delta-1} & \cdots & b_0 \end{bmatrix}}_{\gamma+\delta} \begin{matrix} \left.\vphantom{\begin{matrix}a\\a\\a\end{matrix}}\right\}\delta \\ \left.\vphantom{\begin{matrix}a\\a\\a\end{matrix}}\right\}\gamma \end{matrix}$$

$$\begin{cases} f(x_1, x_2) = 0 \\ g(x_1, x_2) = 0 \end{cases}$$

1. Define system

2. Compute resultant

The Resultant-based Approach

$$\begin{cases} f(x_1, x_2) = 0 \\ g(x_1, x_2) = 0 \end{cases}$$

$$Syl(f,g) = \underbrace{\begin{bmatrix} a_\gamma & \cdots & a_1 & a_0 & & 0 \\ & \ddots & & \ddots & \ddots & \\ 0 & & a_\gamma & \cdots & a_1 & a_0 \\ b_\delta & b_{\delta-1} & \cdots & b_0 & & 0 \\ & \ddots & \ddots & & \ddots & \\ 0 & & b_\delta & b_{\delta-1} & \cdots & b_0 \end{bmatrix}}_{\gamma+\delta} \begin{matrix} \left.\vphantom{\begin{matrix}a\\a\\a\end{matrix}}\right\} \delta \\ \left.\vphantom{\begin{matrix}a\\a\\a\end{matrix}}\right\} \gamma \end{matrix}$$

$$r(x_1) = 0 \rightarrow x_1$$

1. Define system

2. Compute resultant

Univariate root

Summary and complexities

“Basic” system solving (GB, FGLM, univariate solving)

$$\max\left(C(GB), C(FGLM)\right)$$

Estimating $C(GB)$ needs d_{reg}, $C(FGLM)$ is easy

Summary and complexities

"Basic" system solving (GB, FGLM, univariate solving)

$$\max\left(C(GB), C(FGLM)\right)$$

Estimating $C(GB)$ needs d_{reg}, $C(FGLM)$ is easy (until you take e.g. sparcity into account)

Freelunch

- $\max\left(C(\text{Eq gen.}), C(\text{Res})\right)$
- Estimating $C(\text{Eq gen.})$ is hard but doable (and usually not that important)
- There are many different resolution algorithms

Summary and complexities

“Basic” system solving (GB, FGLM, univariate solving)

$$\max\left(C(GB), C(FGLM)\right)$$

Estimating $C(GB)$ needs d_{reg}, $C(FGLM)$ is easy (until you take e.g. sparcity into account)

Freelunch

- $\max\left(C(\mathrm{Eq\ gen.}), C(\mathrm{Res})\right)$
- Estimating $C(\mathrm{Eq\ gen.})$ is hard but doable (and usually not that important)
- There are many different resolution algorithms

Resultants

Only works for a small number of *actual* unknowns, complexity is non-trivial to estimate

Summary and complexities

"Basic" system solving (GB, FGLM, univariate solving)

$$\max\left(C(GB), C(FGLM)\right)$$

Estimating $C(GB)$ needs d_{reg}, $C(FGLM)$ is easy (until you take e.g. sparcity into account)

Freelunch

- $\max\left(C(\mathrm{Eq\ gen.}), C(\mathrm{Res})\right)$
- Estimating $C(\mathrm{Eq\ gen.})$ is hard but doable (and usually not that important)
- There are many different resolution algorithms

Resultants

Only works for a small number of *actual* unknowns, complexity is non-trivial to estimate

Other techniques exist (e.g.Graeffe transform)

These complexities are theoretical

Summary and complexities

“Basic” system solving (GB, FGLM, univariate solving)

$$\max\left(C(GB), C(FGLM)\right)$$

Estimating $C(GB)$ needs d_{reg}, $C(FGLM)$ is easy (until you take e.g. sparcity into account)

Freelunch

- $\max\left(C(\mathrm{Eq\ gen.}), C(\mathrm{Res})\right)$
- Estimating $C(\mathrm{Eq\ gen.})$ is hard but doable (and usually not that important)
- There are many different resolution algorithms

Resultants

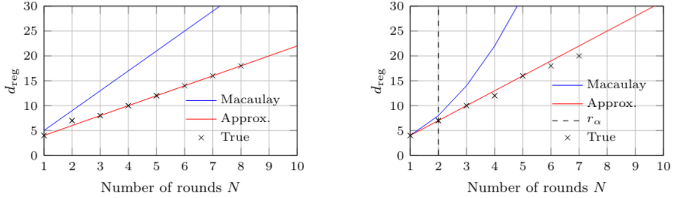
Only works for a small number of *actual* unknowns, complexity is non-trivial to estimate

Other techniques exist (e.g.Graeffe transform)

These complexities are theoretical

In practice, often faster

Example of a Discrepancy



(a) $\mathcal{F}_{\text{CICO}}$. (b) $\mathcal{P}_{\text{CICO}}$.

Figure 6: Theoretical bounds and experimental conjectures for the degree of regularity d_{reg} in Step (1) of a Gröbner basis attack on Anemoi : $\mathbb{F}_p^2 \to \mathbb{F}_p^2$ with $\alpha = 3$. Experimental data points for $p \in \{2^{32} - 209, 2^{64} - 353\}$.

Koschatko, K., Lüftenegger, R., & Rechberger, C. (2024). *Exploring the six worlds of Gröbner basis cryptanalysis: Application to Anemoi.*

IACR Transactions on Symmetric Cryptology, 2024(4), 138-190.

Alternative Title

When POlynomial System SOlving became a threat for symmetric cryptographers

Léo Perrin[1]

[1]Inria, France

ZKCS 2026, February 2026, Vienna

Plan of this Section

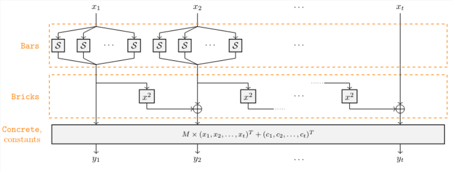
Two Hash Functions vs. Rebound Attacks

Bak, A., Jazeron, G., Galissant, P., & Perrin, L. (2025). **Attacking Split-and-Lookup-Based Primitives Using Probabilistic Polynomial System Solving: Applications to Round-Reduced Monolith and Full-Round Skyscraper**. IACR Transactions on Symmetric Cryptology, 2025(3), 337-367. https://doi.org/10.46586/tosc.v2025.i3.337-367

How to combine S & L and $x \mapsto x^2$ to obtain permutations secure against PoSSo-based attacks and e.g. differential attacks?

Monolith and Skyscraper

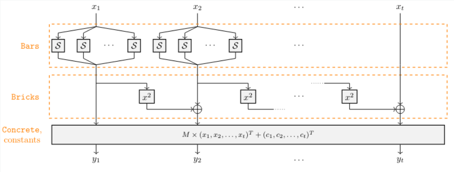
Monolith



- $p \in \{2^{31} - 1, 2^{64} - 2^{32} + 1\}$
- 6 rounds
- S & L ; $x \mapsto x^2$

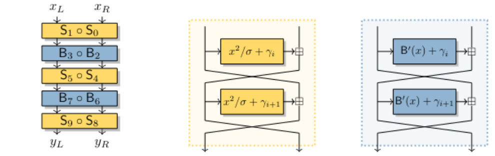
Monolith and Skyscraper

Monolith



- $p \in \{2^{31} - 1, 2^{64} - 2^{32} + 1\}$
- 6 rounds
- S & L ; $x \mapsto x^2$

Skyscraper-v1



(a) Overview of Skyscraper. (b) Details of $S_{i+1} \circ S_i$. (c) Details of $B_{i+1} \circ B_i$.

- $p \geq 2^{256}$
- 10 rounds (but Feistel rounds ≈ count for 1/2)
- S & L ; $x \mapsto x^2$

Attacking Skyscraper-v1

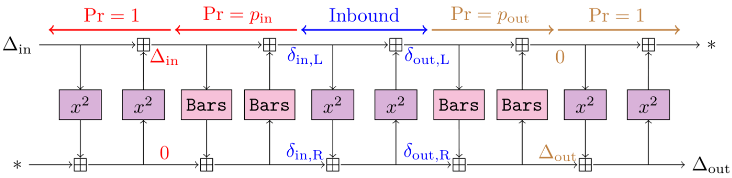


Figure 4: Rebound attack on full Skyscraper-v1.

Table 2: Summary of the main attacks.

Primitive	Attack type	Rounds	Complexity	p_{success}
Skyscraper-v1	Collision (compression)	9	$2^{20.2}$ (practical)	0.9
	Limited birthday	**10**	$2^{8.18}$ (practical)	1

Attacking Monolith (1/2)

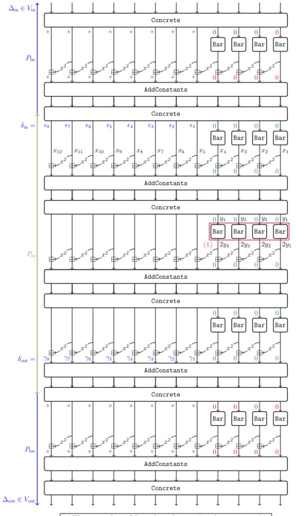


Table 3: Runtime for Gröbner Basis and Change of Order algorithms.

r	Parameters $\log_2 q$	t	k	Gröbner Basis ($\log_2$ #op.)			Change of Order ($\log_2$ #op.)		
4	64	8	4	9.4ω	=	26.38	$6.3\omega + 2$	=	19.68
		12	8	20ω	=	56.14	$12.6\omega + 3$	=	38.36
	31	16	8	20ω	=	56.14	$12.6\omega + 3$	=	38.36
		24	16	41.6ω	=	116.77	$25\omega + 4$	=	74.17
	64	12	≤ 7	20ω	=	56.14	$12.6\omega + 3$	=	38.36
		8	≤ 3	9.4ω	=	26.38	$6.3\omega + 2$	=	19.68
	31	24	≤ 15	41.6ω	=	116.77	$25\omega + 4$	=	74.17
		16	≤ 7	20ω	=	56.14	$12.6\omega + 3$	=	38.36
5	64	12	4	65.3ω	=	183.29	$39\omega + 4.2$	=	111.47
	31	24	8	135.6ω	=	380.62	$79\omega + 5.3$	=	227.05

Table 4: In blue, upper bound from estimations. In the last column, the tuples represent the different possibilities for $(N, \mathbb{P}(\mathrm{NS} = 0))$ where N is the overhead (number of PoSSo instances to solve).

r	Parameters $\log_2 q$	t	k	Assumptions proba. Differential	Linear	(# PoSSo, failure probability) trade-offs
4	64	8	4	p_i	$2^{-22.24}$	$(2^{31}, 2^{-4.9})$; $(2^{34}, 2^{-11.5})$; $(2^{37}, 2^{-21.4})$
		12	8			
	31	16	8	p_i	$2^{-12.64}$	$(2^{18}, 2^{-5.3})$; $(2^{20}, 2^{-10.2})$; $(2^{22}, 2^{-17})$
		24	16			
	64	12	≤ 7	1	1	$(1, 0)$
	31	24	≤ 15			
	64	8	≤ 3	p_i	1	$(50, 2^{-19.7})$
	31	16	≤ 7			$(80, 2^{-20.4})$

Attacking Monolith (2/2)

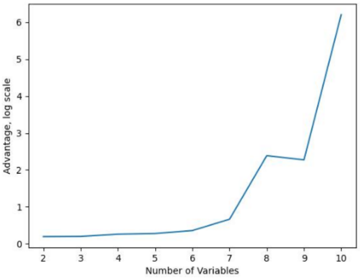


Figure 10: Plot of logarithm in base two of the mean advantage for order lex.

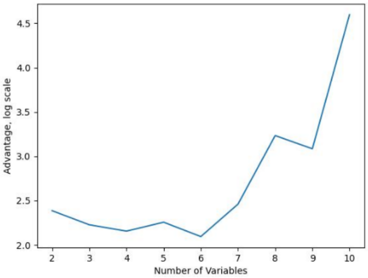


Figure 11: Plot of the logarithm in base two of the mean advantage for order degrevlex.

Plan of this Section

What have we learnt?

1. Many cryptanalyses become possible thanks to PoSSo (CICO-1 of course, but also rebound, key recovery...)

What have we learnt?

1. Many cryptanalyses become possible thanks to PoSSo (CICO-1 of course, but also rebound, key recovery...)
2. The exact complexity of the PoSSo step remains mysterious...

What have we learnt?

1. Many cryptanalyses become possible thanks to PoSSo (CICO-1 of course, but also rebound, key recovery...)
2. The exact complexity of the PoSSo step remains mysterious...
3. ... and anyway this step can be approached in many different ways

What have we learnt?

1. Many cryptanalyses become possible thanks to PoSSo (CICO-1 of course, but also rebound, key recovery...)
2. The exact complexity of the PoSSo step remains mysterious...
3. ... and anyway this step can be approached in many different ways
4. A piori sensible design strategies can fail completely (Skyscraper-V1, Anemoi...)

What do symmetric cryptographers need? (IMO)

1. Better ressources **for symmetric cryptographers** on PoSSo
   - What are the relevant algorithms? Their complexities?
   - What are potentially interesting papers/techniques in the PoSSO literature?

What do symmetric cryptographers need? (IMO)

1. Better ressources **for symmetric cryptographers** on PoSSo
   - What are the relevant algorithms? Their complexities?
   - What are potentially interesting papers/techniques in the PoSSO literature?

SoK? Wiki? "living" survey on github?

What do symmetric cryptographers need? (IMO)

1. Better ressources **for symmetric cryptographers** on PoSSo
   - What are the relevant algorithms? Their complexities?
   - What are potentially interesting papers/techniques in the PoSSO literature?

   SoK? Wiki? "living" survey on github?

2. Better software **for symmetric cryptographers** on PoSSo
   - Easily test different solving technique
   - Implement sophisticated state-of-the-art algorithms

PESSCY conclusion

README

Polynomial Equations Solving for Symmetric Cryptography (PESSCY)

PESSCY is a tool developed to allow practical analysis of algebraic attacks using the Gröbner basis theory over Oriented-Arithmetisation Primitives.

Installation of the tool

This tool must be used and installed in an environment containing SageMath.

To install it, run in the folder where you clone this repository:

```
pip install -e .
```

https://github.com/bbdaumen/PESSCY

(by Baptiste Daumen)

PESSCY conclusion

README

Polynomial Equations Solving for Symmetric Cryptography (PESSCY)

PESSCY is a tool developed to allow practical analysis of algebraic attacks using the Gröbner basis theory over Oriented-Arithmetisation Primitives.

Installation of the tool

This tool must be used and installed in an environment containing SageMath.

To install it, run in the folder where you clone this repository:

```
pip install -e .
```

https://github.com/bbdaumen/PESSCY

(by Baptiste Daumen)

Thank You!