What happens when you **don't** have tools? The case of algebraic cryptanalysis

Léo Perrin[1]

[1]Inria, France

March 15, 2025

In this Presentation

A *tool* is a convenient abstraction layer above a specific know-how.

In this Presentation

A *tool* is a convenient abstraction layer above a specific know-how.

This comes not from my satisfaction with some tools...

In this Presentation

A *tool* is a convenient abstraction layer above a specific know-how.

This comes not from my satisfaction with some tools...

... but from their absence*!

* at least at the time when it happened

Outline

Plan of this Section

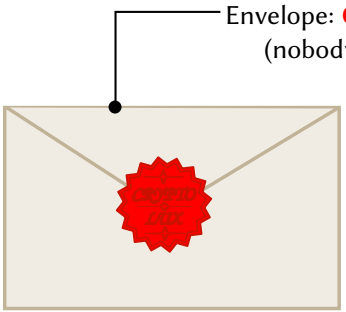
Plan of this Section

Securing Data

Usually, we secure data (at rest or in transit).

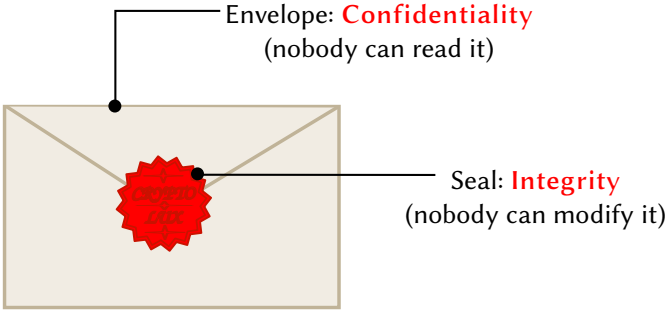
Securing Data

Usually, we secure data (at rest or in transit).



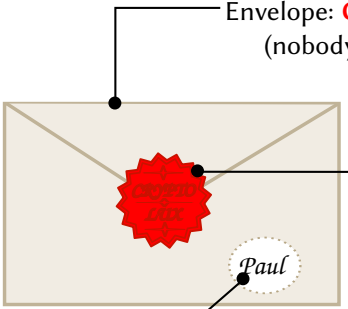
Securing Data

Usually, we secure data (at rest or in transit).



Securing Data

Usually, we secure data (at rest or in transit).



Securing Computation

More and more protocols intend to secure computations.

FHE Fully Homomorphic Encryption

MPC Multi Party Computations

ZK-* Zero Knowledge- [proof, argument...]

Securing Computation

More and more protocols intend to secure computations.

FHE Fully Homomorphic Encryption

MPC Multi Party Computations

ZK-* Zero Knowledge- [proof, argument...]

These need (many) different types of symmetric primitives internally, and these are *often* "Arithmetization-Oriented".

What is Arithmetization?

"Arithmetization" depends on the subtleties of the protocol you work with!

What is Arithmetization?

"Arithmetization" depends on the subtleties of the protocol you work with!

Example: Verifying if $y = c(ax + b)^{10} + x$ in R1CS

1. $t_0 = ax$
2. $t_1 = t_0 + b$
3. $t_2 = t_1 \times t_1$
4. $t_3 = t_2 \times t_2$
5. $t_4 = t_3 \times t_3$
6. $t_5 = t_2 \times t_4$
7. $t_6 = ct_5$
8. $y = t_6 + x$

What is Arithmetization?

"Arithmetization" depends on the subtleties of the protocol you work with!

Example: Verifying if $y = c(ax + b)^{10} + x$ in R1CS

1. $t_0 = ax$
2. $t_1 = t_0 + b$
3. $t_2 = t_1 \times t_1$
4. $t_3 = t_2 \times t_2$
5. $t_4 = t_3 \times t_3$
6. $t_5 = t_2 \times t_4$
7. $t_6 = ct_5$
8. $y = t_6 + x$

What is Arithmetization?

"Arithmetization" depends on the subtleties of the protocol you work with!

Example: Verifying if $y = c(ax + b)^{10} + x$ in R1CS

1. $t_0 = ax$
2. $t_1 = t_0 + b$
3. $t_2 = t_1 \times t_1$
4. $t_3 = t_2 \times t_2$
5. $t_4 = t_3 \times t_3$
6. $t_5 = t_2 \times t_4$
7. $t_6 = ct_5$
8. $y = t_6 + x$

This verification costs R1CS 4 constaints

What is Arithmetization?

"Arithmetization" depends on the subtleties of the protocol you work with!

Example: Verifying if $y = c(ax + b)^{10} + x$ in R1CS

1. $t_0 = ax$
2. $t_1 = t_0 + b$
3. $t_2 = t_1 \times t_1$
4. $t_3 = t_2 \times t_2$
5. $t_4 = t_3 \times t_3$
6. $t_5 = t_2 \times t_4$
7. $t_6 = ct_5$
8. $y = t_6 + x$

This verification costs R1CS 4 constaints

- How to turn a computation into an arithmetic circuit depends on the operations allowed
- Its cost is also arithmetization-dependent—though low degree is usually welcome!

Symmetric Techniques for Advanced Protocols

MPC

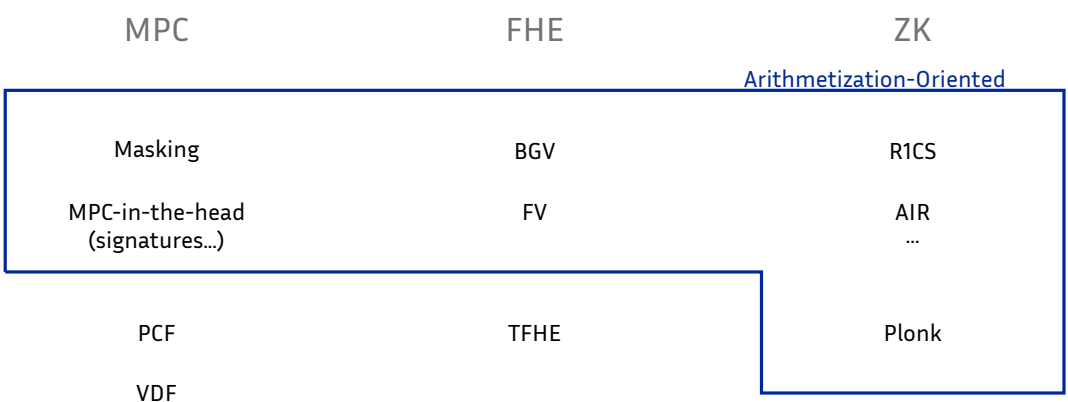
FHE

ZK

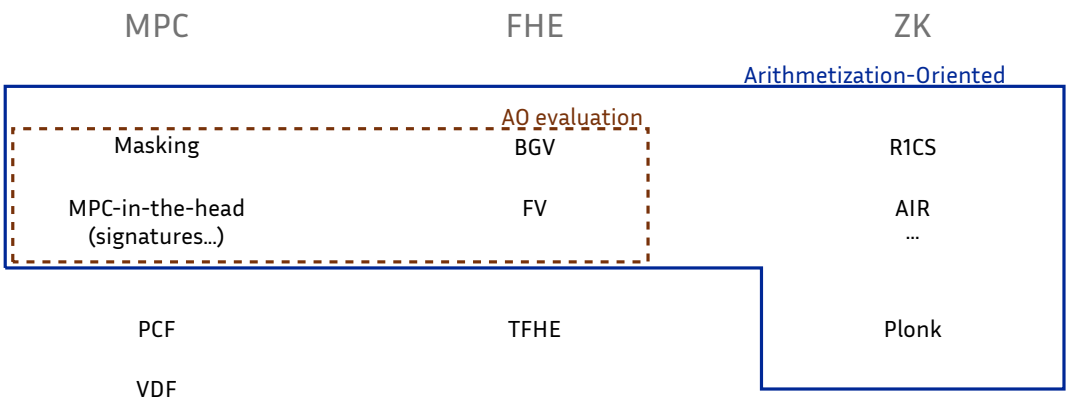
Symmetric Techniques for Advanced Protocols

MPC	FHE	ZK
Masking	BGV	R1CS
MPC-in-the-head (signatures...)	FV	AIR ...
PCF	TFHE	Plonk
VDF		

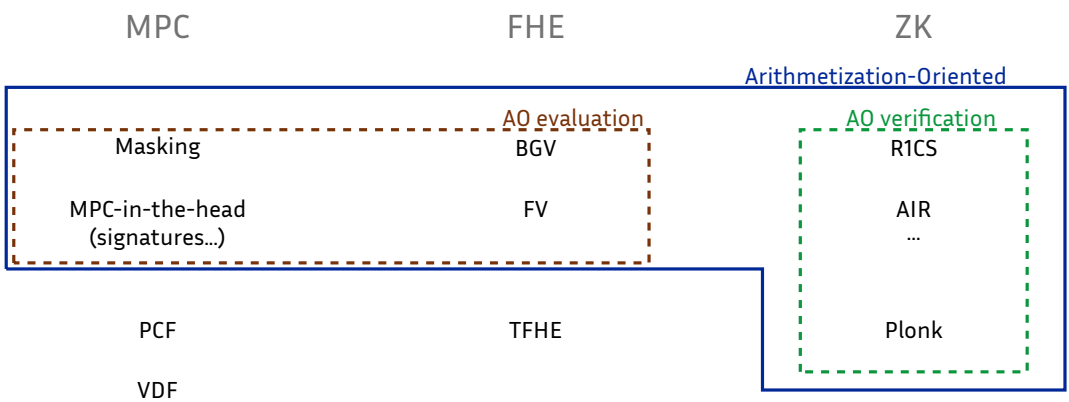
Symmetric Techniques for Advanced Protocols



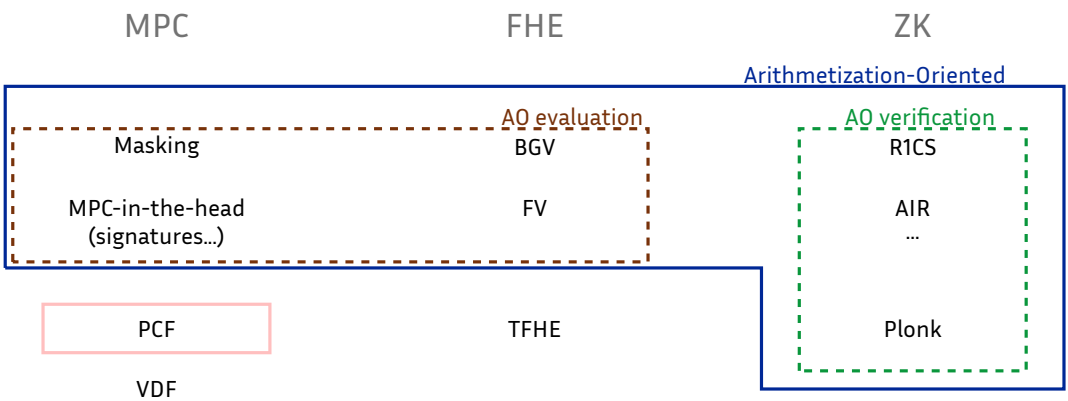
Symmetric Techniques for Advanced Protocols



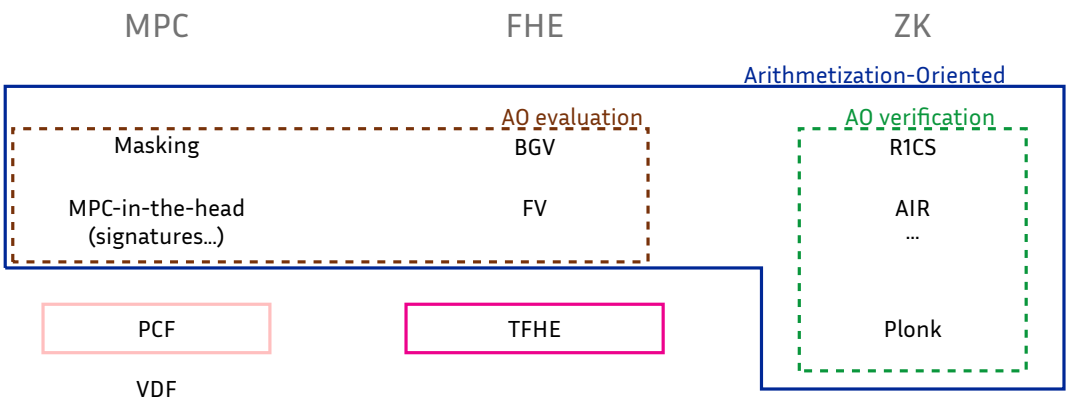
Symmetric Techniques for Advanced Protocols



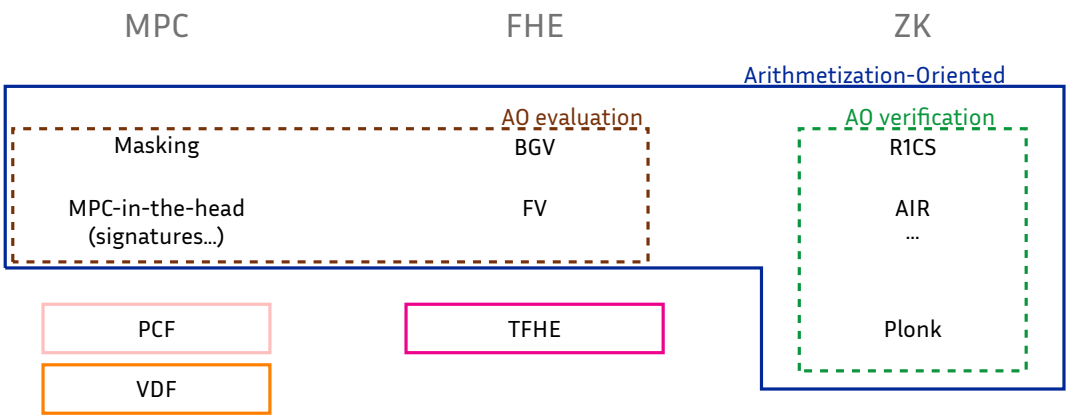
Symmetric Techniques for Advanced Protocols



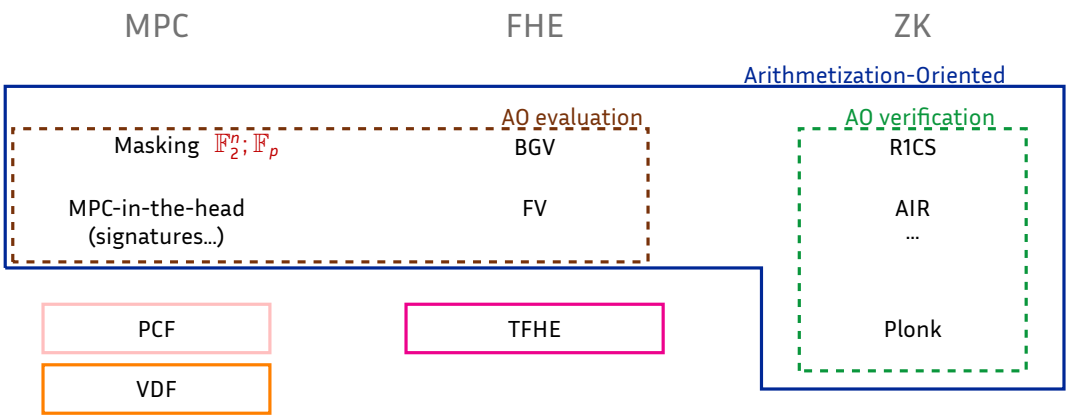
Symmetric Techniques for Advanced Protocols



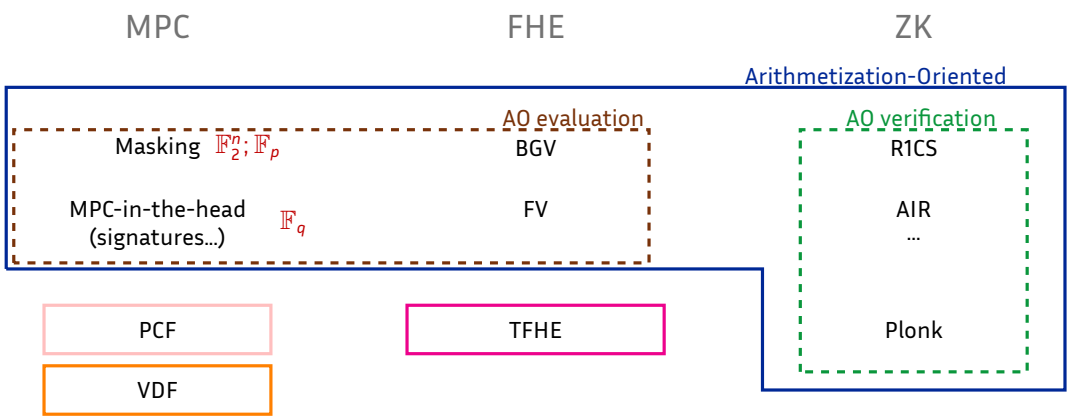
Symmetric Techniques for Advanced Protocols



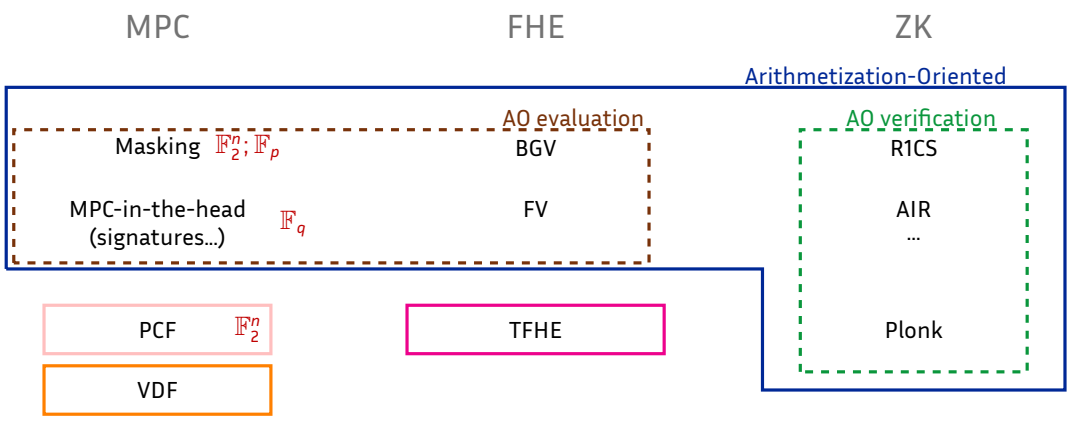
Symmetric Techniques for Advanced Protocols



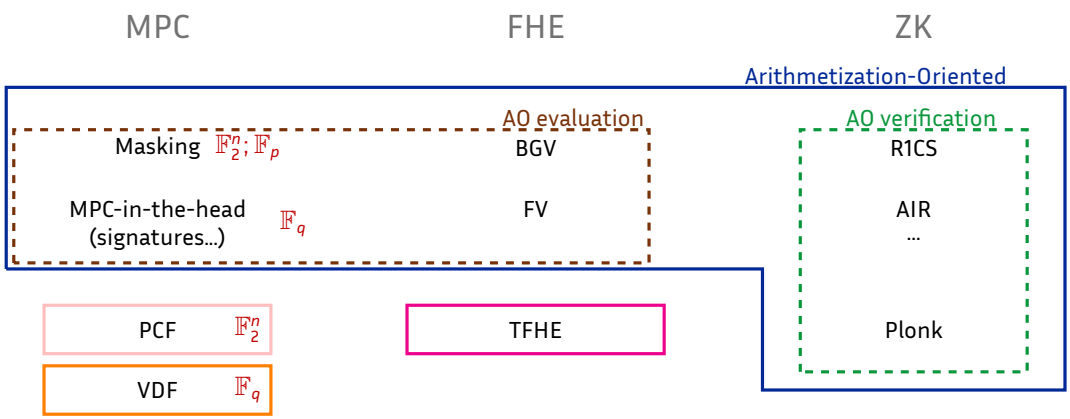
Symmetric Techniques for Advanced Protocols



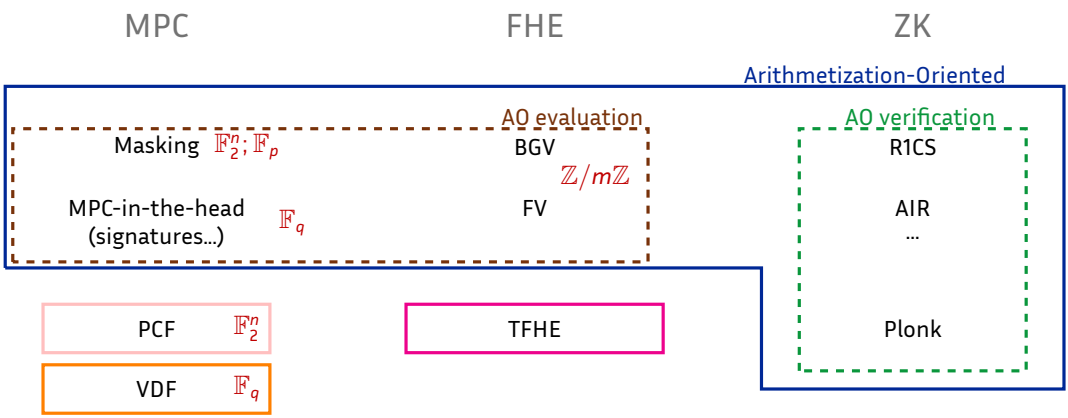
Symmetric Techniques for Advanced Protocols



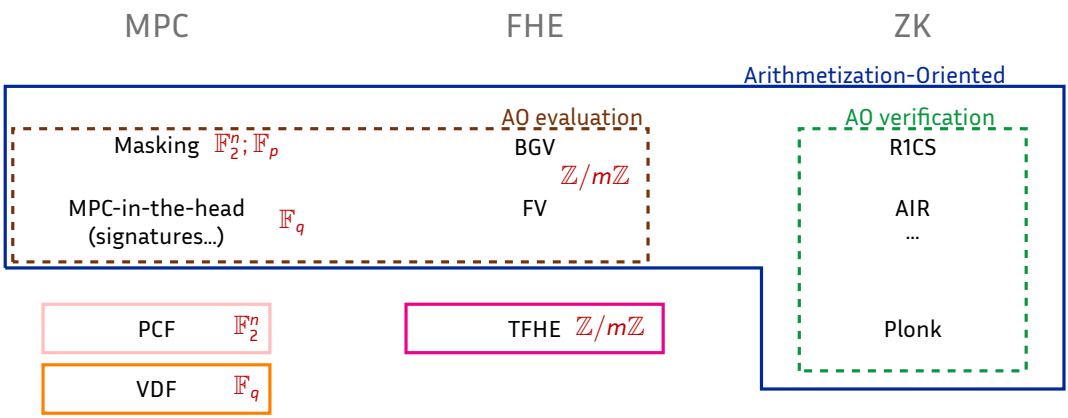
Symmetric Techniques for Advanced Protocols



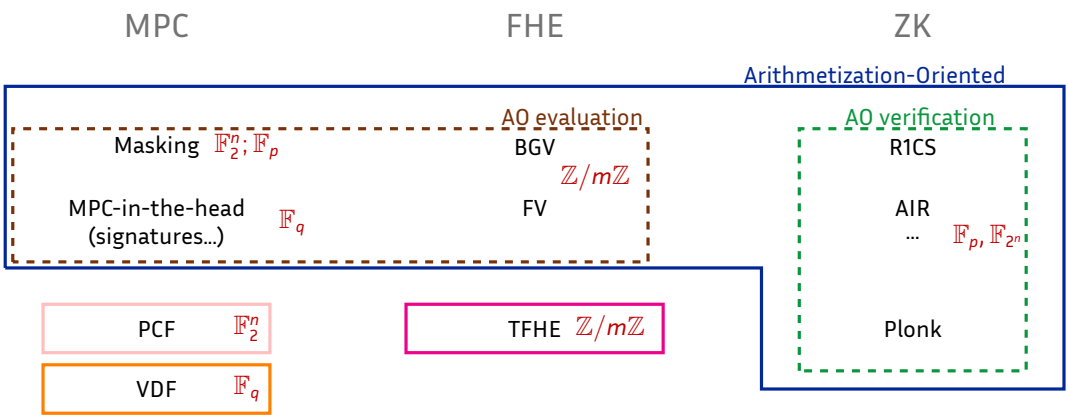
Symmetric Techniques for Advanced Protocols



Symmetric Techniques for Advanced Protocols



Symmetric Techniques for Advanced Protocols



Arithmetization-Oriented Designs

What AO Often Means

The evaluation or the verification of the subfunctions involved must correspond to an arithmetic circuit with a low number of multiplications.

Arithmetization-Orientated Designs

What AO Often Means

The evaluation or the verification of the subfunctions involved must correspond to an arithmetic circuit with a low number of multiplications.

Disclaimer: *this is a massive over simplification*

Arithmetization-Orientated Designs

What AO Often Means

The evaluation or the verification of the subfunctions involved must correspond to an arithmetic circuit with a low number of multiplications.

Disclaimer: *this is a massive over simplification*

- Use of $x \mapsto x^d$, d "small"
- Big diffusion matrices
- For verif., $x \mapsto x^{1/d}$, d "small"

Arithmetization-Oriented Designs

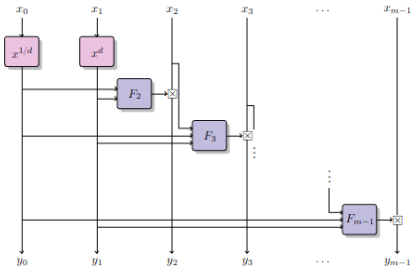
What AO Often Means

The evaluation or the verification of the subfunctions involved must correspond to an arithmetic circuit with a low number of multiplications.

Disclaimer: *this is a massive over simplification*

- Use of $x \mapsto x^d$, d "small"
- Big diffusion matrices
- For verif., $x \mapsto x^{1/d}$, d "small"

Example: Griffin



Source: PhD thesis of C. Bouvier, *Cryptanalysis and design of symmetric primitives defined over large finite fields*

Plan of this Section

Vulnerability to Algebraic Attacks

Arithmetization-Oriented Verification

Vulnerability to Algebraic Attacks

Arithmetization-Oriented Verification

There must exist low degree arithmetic circuits that the successive internal states must satisfy.

Vulnerability to Algebraic Attacks

Arithmetization-Oriented Verification

There must exist low degree arithmetic circuits that the successive internal states must satisfy.

Vulnerability to Algebraic Attacks

Vulnerability to Algebraic Attacks

Arithmetization-Oriented Verification

There must exist low degree arithmetic circuits that the successive internal states must satisfy.

Vulnerability to Algebraic Attacks

There must exist a system of low degree equations that models an evaluation of the primitive.

Vulnerability to Algebraic Attacks

Arithmetization-Oriented Verification

There must exist low degree arithmetic circuits that the successive internal states must satisfy.

$\Longrightarrow$

Vulnerability to Algebraic Attacks

There must exist a system of low degree equations that models an evaluation of the primitive.

Vulnerability to Algebraic Attacks

Arithmetization-Oriented Verification

There must exist low degree arithmetic circuits that the successive internal states must satisfy.

$\Longrightarrow$

Vulnerability to Algebraic Attacks

There must exist a system of low degree equations that models an evaluation of the primitive.

Enabling a low degree arithmetization of verification implies a potential vulnerability to algebraic attacks!!

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** slooooow

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...

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

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

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

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

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... **but** lots of variants with very different complexities

Generic System Solving: Steps and Tools

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

1. Define system

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

2. Find a GB (F4/F5)

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

3. Change order to **lex**

$$g_N^*(x_N) = 0 \rightarrow x_N$$

4. Univariate root

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... **but** lots of variants with very different complexities

Berlekamp-Rabin Easy to re-implement

The Freelunch Approach

Steps 2 and 3 are both computationally costly, but not for the same reasons. For most AOPs, step 2 dominates in practice...

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

1. Define system
2. Find a GB
3. Change order to **lex**
4. Univariate root

The Freelunch Approach

Steps 2 and 3 are both computationally costly, but not for the same reasons. For most AOPs, step 2 dominates in practice...

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

1. Define system

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

2. Find a GB

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

3. Change order to **lex**

$$g_N^*(x_N) = 0 \rightarrow x_N$$

4. Univariate root

We can skip it!

Freelunch Crytpanalysis: Results

https://eprint.iacr.org/2024/347

Bariant, Augustin, et al. *The algebraic FreeLunch: Efficient Gröbner basis attacks against arithmetization-oriented primitives.* Annual International Cryptology Conference. Cham: Springer Nature Switzerland, 2024.

Name	α/e	Number of branches						
		2	3	4	5	6	8	≥ 12
Griffin	3	$\emptyset$	120 (16)	112 (15)	$\emptyset$	$\emptyset$	76 (11)	64 (10)
	5	$\emptyset$	141 (14)	110 (11)	$\emptyset$	$\emptyset$	81 (9)	74 (9)
Arion	3	$\emptyset$	128 (6)	134 (6)	114 (5)	119 (5)	98 (4)	$\emptyset$
	5	$\emptyset$	132 (6)	113 (5)	118 (5)	122 (5)	101 (4)	$\emptyset$
α-Arion	3	$\emptyset$	104 (5)	84 (4)	88 (4)	92 (4)	98 (4)	$\emptyset$
	5	$\emptyset$	83 (4)	87 (4)	91 (4)	94 (4)	101 (4)	$\emptyset$
Anemoi	3	118 (21)	$\emptyset$	-	$\emptyset$	-	-	-
	5	156 (21)	$\emptyset$	-	$\emptyset$	-	-	-
	7	174 (20)	$\emptyset$	-	$\emptyset$	-	-	-
	11	198 (19)	$\emptyset$	-	$\emptyset$	-	-	-

Plan of this Section

Post-Mortem

From the designers' perspective

SAGE Did what it was supposed to do

Post-Mortem

From the designers' perspective

SAGE Did what it was supposed to do

F4/F5 **Turned out to be entirely irrelevant**

Post-Mortem

From the designers' perspective

SAGE Did what it was supposed to do

F4/F5 **Turned out to be entirely irrelevant**

FGLM **Was replaced by a dedicated variant**

Post-Mortem

From the designers' perspective

SAGE	Did what it was supposed to do
F4/F5	**Turned out to be entirely irrelevant**
FGLM	**Was replaced by a dedicated variant**
Berlekamp-Rabin	Did what it was supposed to do

Post-Mortem

From the designers' perspective

SAGE Did what it was supposed to do

F4/F5 **Turned out to be entirely irrelevant**

FGLM **Was replaced by a dedicated variant**

Berlekamp-Rabin Did what it was supposed to do

- Attackers didn't improve the tools, they created new ones

Post-Mortem

From the designers' perspective

SAGE Did what it was supposed to do

F4/F5 **Turned out to be entirely irrelevant**

FGLM **Was replaced by a dedicated variant**

Berlekamp-Rabin Did what it was supposed to do

- Attackers didn't improve the tools, they created new ones
  in particular, a technique to build dedicated monomial orderings

Post-Mortem

From the designers' perspective

SAGE Did what it was supposed to do

F4/F5 **Turned out to be entirely irrelevant**

FGLM **Was replaced by a dedicated variant**

Berlekamp-Rabin Did what it was supposed to do

- Attackers didn't improve the tools, they created new ones
  in particular, a technique to build dedicated monomial orderings
- Designers' relied on the assumption that the F4/F5 $\longrightarrow$ FGLM $\longrightarrow$ Univariate root finding chain was the **only** multivariate root-finding tool.

Post-Mortem

From the designers' perspective

SAGE Did what it was supposed to do

F4/F5 **Turned out to be entirely irrelevant**

FGLM **Was replaced by a dedicated variant**

Berlekamp-Rabin Did what it was supposed to do

- Attackers didn't improve the tools, they created new ones
  in particular, a technique to build dedicated monomial orderings
- Designers' relied on the assumption that the F4/F5 $\longrightarrow$ FGLM $\longrightarrow$ Univariate root finding chain was the **only** multivariate root-finding tool.
- Inefficient/proprietary software limited the designers' ability to prototype attacks.

What Did We Need?

A better ability to prototype

A better interface with another field

What Did We Need?

A better ability to prototype

- Derive systems of equations corresponding to different modeling

A better interface with another field

What Did We Need?

A better ability to prototype

- Derive systems of equations corresponding to different modeling
- Test in practical time the efficiency of different algorithms

A better interface with another field

What Did We Need?

A better ability to prototype

- Derive systems of equations corresponding to different modeling
- Test in practical time the efficiency of different algorithms

A better interface with another field

- Underlying assumptions when using the F4/F5 $\longrightarrow$ FGLM $\longrightarrow$ Univariate root finding

What Did We Need?

A better ability to prototype

- Derive systems of equations corresponding to different modeling
- Test in practical time the efficiency of different algorithms

A better interface with another field

- Underlying assumptions when using the F4/F5 $\longrightarrow$ FGLM $\longrightarrow$ Univariate root finding
- Deeper understanding of each step

What Did We Need?

A better ability to prototype

- Derive systems of equations corresponding to different modeling
- Test in practical time the efficiency of different algorithms

A better interface with another field

- Underlying assumptions when using the F4/F5 $\longrightarrow$ FGLM $\longrightarrow$ Univariate root finding
- Deeper understanding of each step

We needed better tools!

Plan of this Section

Plan of this Section

Lessons from the MILP and Root Finding Case

Root Finding

- The chain F4/F5 $\longrightarrow$ FGLM $\longrightarrow$ Univariate root finding is great for generic systems, but ours are very structured
- Finer grained knowledge allowed the freelunch attacks...

Lessons from the MILP and Root Finding Case

Root Finding

- The chain F4/F5 $\longrightarrow$ FGLM $\longrightarrow$ Univariate root finding is great for generic systems, but ours are very structured
- Finer grained knowledge allowed the freelunch attacks...
- ... and will allow better security arguments

Lessons from the MILP and Root Finding Case

Root Finding

- The chain F4/F5 $\longrightarrow$ FGLM $\longrightarrow$ Univariate root finding is great for generic systems, but ours are very structured
- Finer grained knowledge allowed the freelunch attacks...
- ... and will allow better security arguments

MILP

Questions for the audience:

Do we always use the "standard" solving method? Have we looked into a deeper integration?

Lessons from the MILP and Root Finding Case

Root Finding

- The chain F4/F5 $\longrightarrow$ FGLM $\longrightarrow$ Univariate root finding is great for generic systems, but ours are very structured
- Finer grained knowledge allowed the freelunch attacks...
- ... and will allow better security arguments

MILP

Questions for the audience:

Do we always use the "standard" solving method? Have we looked into a deeper integration?

We can't be experts in all fields: we need a way to "hide" some of the complexity to make it usable, while understanding enough to find optimization directions.

Lessons from the MILP and Root Finding Case

Root Finding

- The chain F4/F5 $\longrightarrow$ FGLM $\longrightarrow$ Univariate root finding is great for generic systems, but ours are very structured
- Finer grained knowledge allowed the freelunch attacks...
- ... and will allow better security arguments

MILP

Questions for the audience:

Do we always use the "standard" solving method? Have we looked into a deeper integration?

We can't be experts in all fields: we need a way to "hide" some of the complexity to make it usable, while understanding enough to find optimization directions.

There is a "correct" level of abstraction that we must find

The Purposes of Tools

When do we talk about “tools”?

The Purposes of Tools

When do we talk about “tools”?

1. MILP is a convenient tool to study differential trails.

The Purposes of Tools

When do we talk about “tools”?

1. MILP is a convenient tool to study differential trails.
2. Ideally, we should have tools that can convincingly say “this is primitive is secure”

The Purposes of Tools

When do we talk about “tools”?

1. MILP is a convenient tool to study differential trails.
2. Ideally, we should have tools that can convincingly say “this is primitive is secure”
3. The low performance of some tools prevented primitive designers from prototyping attacks

The Purposes of Tools

When do we talk about “tools”?

1. MILP is a convenient tool to study differential trails.
2. Ideally, we should have tools that can convincingly say “this is primitive is secure”
3. The low performance of some tools prevented primitive designers from prototyping attacks
4. We wished the tools to solve multivariate root finding problems were easier (for us!) to understand and use.

The Purposes of Tools

When do we talk about "tools"?

1. MILP is a convenient tool to study differential trails.
2. Ideally, we should have tools that can convincingly say "this is primitive is secure"
3. The low performance of some tools prevented primitive designers from prototyping attacks
4. We wished the tools to solve multivariate root finding problems were easier (for us!) to understand and use.

That's a lot "tools"! And yet...

The Purposes of Tools

When do we talk about "tools"?

1. MILP is a convenient tool to study differential trails.
2. Ideally, we should have tools that can convincingly say "this is primitive is secure"
3. The low performance of some tools prevented primitive designers from prototyping attacks
4. We wished the tools to solve multivariate root finding problems were easier (for us!) to understand and use.

That's a lot "tools"! And yet...

Definition ("Tool")

A *tool* is a convenient abstraction layer above a specific know-how.

Plan of this Section

A Prototyping Wish-List

I wish I had open source and efficient tools to...

A Prototyping Wish-List

I wish I had open source and efficient tools to...

- Generate (and test!) a modeling of a round function using non-linear equations

A Prototyping Wish-List

I wish I had open source and efficient tools to...

- Generate (and test!) a modeling of a round function using non-linear equations
- Handle fields of many different sizes

A Prototyping Wish-List

I wish I had open source and efficient tools to...

- Generate (and test!) a modeling of a round function using non-linear equations
- Handle fields of many different sizes
- Manipulate multivariate polynomials efficiently: arithmetic, composition, reduction, different monomial orderings...

A Prototyping Wish-List

I wish I had open source and efficient tools to...

- Generate (and test!) a modeling of a round function using non-linear equations
- Handle fields of many different sizes
- Manipulate multivariate polynomials efficiently: arithmetic, composition, reduction, different monomial orderings...
- Estimate the complexity of some “standard” algorithms: FGLM? F4/F5?

A Prototyping Wish-List

I wish I had open source and efficient tools to...

- Generate (and test!) a modeling of a round function using non-linear equations
- Handle fields of many different sizes
- Manipulate multivariate polynomials efficiently: arithmetic, composition, reduction, different monomial orderings...
- Estimate the complexity of some "standard" algorithms: FGLM? F4/F5?
- Suggest the best known algorithm for some problems

MIDC: Towards Sound Security Arguments?

The resolution of a multivariate system of equations can be approached in many ways, but the ideal degree is an inherent property of the system

$\rightarrow$ a sounder basis for security arguments!

MIDC: Towards Sound Security Arguments?

The resolution of a multivariate system of equations can be approached in many ways, but the ideal degree is an inherent property of the system

$\rightarrow$ a sounder basis for security arguments!

Monotonous Ideal Degree Conjecture

A method to find a reasonable lower bound for the ideal degree.

`https://eprint.iacr.org/2024/605`

$$\left\{\begin{array}{l}\left.\begin{array}{l} x_0^{d_0} - P'_0(x_0, \ldots, x_{n-1}) \\ x_1^{d_1} - P'_1(x_0, \ldots, x_{n-1}) \\ \quad\ldots \\ x_{b-1}^{d_{b-1}} - P'_{b-1}(x_0, \ldots, x_{n-1}) \end{array}\right\} \text{basal part} \\ \\ P_b(x_0, \ldots, x_{n-1}) \\ P_{b+1}(x_0, \ldots, x_{n-1}) \\ \quad\ldots \\ P_{n-1}(x_0, \ldots, x_{n-1}) \end{array}\right.$$

Plan of this Section

Conclusion

What is a Tool?

IMHO: *a convenient layer of abstraction above a specific know-how.*

Past Issues with Algebraic Attacks

- No convenient approach to estimate attack complexity

Conclusion

What is a Tool?

IMHO: *a convenient layer of abstraction above a specific know-how.*

Past Issues with Algebraic Attacks

- No convenient approach to estimate attack complexity
- No convenient software tooling to write down relevant equations

Conclusion

What is a Tool?

IMHO: *a convenient layer of abstraction above a specific know-how.*

Past Issues with Algebraic Attacks

- No convenient approach to estimate attack complexity
- No convenient software tooling to write down relevant equations
- No convenient software to solve them (SAGE -> Magma)

Conclusion

What is a Tool?

IMHO: *a convenient layer of abstraction above a specific know-how.*

Past Issues with Algebraic Attacks

- No convenient approach to estimate attack complexity
- No convenient software tooling to write down relevant equations
- No convenient software to solve them (SAGE -> Magma)

... But there is hope!

- the MIDC could be used to build security arguments,
- future software tools will ease prototyping of algebraic attacks!

Conclusion

What is a Tool?

IMHO: *a convenient layer of abstraction above a specific know-how.*

Past Issues with Algebraic Attacks

- No convenient approach to estimate attack complexity
- No convenient software tooling to write down relevant equations
- No convenient software to solve them (SAGE -> Magma)

... But there is hope!

- the MIDC could be used to build security arguments,
- future software tools will ease prototyping of algebraic attacks!

Thank you!