Why do all these primitives defined over rings/prime fields get broken?

Léo Perrin[1]

[1]Inria, France

SPRING/EC'26/CAHF

Full Round Attacks

Full Round Attacks

ZK-friendly hash functions

`Jarvis` (2018), `Griffin` (2022), `Anemoi` (2022), `Arion` (2023), and `Skyscrapper v1` (2025)

Full Round Attacks

ZK-friendly hash functions

`Jarvis` (2018), `Griffin` (2022), `Anemoi` (2022), `Arion` (2023), and `Skyscrapper v1` (2025)

- `Skyscraper` fell to a rebound attack
- **All others** fell to PoSSo-based attacks

Full Round Attacks

ZK-friendly hash functions

`Jarvis` (2018), `Griffin` (2022), `Anemoi` (2022), `Arion` (2023), and `Skyscrapper v1` (2025)

- `Skyscraper` fell to a rebound attack
- **All others** fell to PoSSo-based attacks

FHE-friendly stream ciphers

`RASTA` (2018), `DASTA` (2020), `Elisabeth-4` (2022) and `FRAST` (2024)

Full Round Attacks

ZK-friendly hash functions

Jarvis (2018), Griffin (2022), Anemoi (2022), Arion (2023), and Skyscrapper v1 (2025)

- Skyscraper fell to a rebound attack
- **All others** fell to PoSSo-based attacks

FHE-friendly stream ciphers

RASTA (2018), DASTA (2020), Elisabeth-4 (2022) and FRAST (2024)

- [RD]ASTA and Elisabeth-4 fell to linearization attacks

Full Round Attacks

ZK-friendly hash functions

`Jarvis` (2018), `Griffin` (2022), `Anemoi` (2022), `Arion` (2023), and `Skyscrapper v1` (2025)

- `Skyscraper` fell to a rebound attack
- **All others** fell to PoSSo-based attacks

FHE-friendly stream ciphers

`RASTA` (2018), `DASTA` (2020), `Elisabeth-4` (2022) and `FRAST` (2024)

- `[RD]ASTA` and `Elisabeth-4` fell to linearization attacks
- `FRAST` relies on a blockcipher with probability 1 iterated differentials

Full Round Attacks

ZK-friendly hash functions

`Jarvis` (2018), `Griffin` (2022), `Anemoi` (2022), `Arion` (2023), and `Skyscrapper v1` (2025)

- `Skyscraper` fell to a rebound attack
- **All others** fell to PoSSo-based attacks

FHE-friendly stream ciphers

`RASTA` (2018), `DASTA` (2020), `Elisabeth-4` (2022) and `FRAST` (2024)

- `[RD]ASTA` and `Elisabeth-4` fell to linearization attacks
- `FRAST` relies on a blockcipher with probability 1 iterated differentials

What happened?

Full Round Attacks

ZK-friendly hash functions

`Jarvis` (2018), `Griffin` (2022), `Anemoi` (2022), `Arion` (2023), and `Skyscrapper v1` (2025)

- `Skyscraper` fell to a rebound attack
- **All others** fell to PoSSo-based attacks

FHE-friendly stream ciphers

`RASTA` (2018), `DASTA` (2020), `Elisabeth-4` (2022) and `FRAST` (2024)

- `[RD]ASTA` and `Elisabeth-4` fell to linearization attacks
- `FRAST` relies on a blockcipher with probability 1 iterated differentials

What happened? **Should we PANIC?!??**

Outline

1 Full round attacks

2 These primitives are harder to design

3 Causes of death

4 Conclusion

Plan of this Section

1 Full round attacks

2 These primitives are harder to design

3 Causes of death

4 Conclusion

Plan of this Section

1 Full round attacks

2 These primitives are harder to design
- On designing a symmetric primitive
- New use cases: the need for arithmetizations
- STAP Examples

3 Causes of death

4 Conclusion

A Question

Why do we use symmetric primitives?

A Question

Why do we use symmetric primitives?

Security · Efficiency

Unstable Definitions

What is “efficient” varies

- What are the operations that we **can** use?
- What are the associated **costs**?

How to get the best security for a given price?

What is “secure” varies

How do we define the **security** that the primitive must provide?

Unstable Definitions

What is “efficient” varies

- What are the operations that we **can** use?
- What are the associated **costs**?

How to get the best security for a given price?

What is “secure” varies

How do we define the **security** that the primitive must provide?

What are the relevant forms of cryptanalysis?

Unstable Definitions

What is “efficient” varies

- What are the operations that we **can** use?
- What are the associated **costs**?

How to get the best security for a given price?

What is “secure” varies

How do we define the **security** that the primitive must provide?

What are the relevant forms of cryptanalysis? (We’ll get back to that later)

Plan of this Section

1 Full round attacks

2 These primitives are harder to design
- On designing a symmetric primitive
- New use cases: the need for arithmetizations
- STAP Examples

3 Causes of death

4 Conclusion

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...]

One Approach to Rule Them All (?): arithmetization

A Basic Example of Arithmetization

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

A Basic Example of Arithmetization

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

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. $y = t_6 + x$

A Basic Example of Arithmetization

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

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. $y = t_6 + x$

A Basic Example of Arithmetization

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

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

A Basic Example of Arithmetization

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

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!

A not basic at all example of arithmetization

The cost of each operation depends on the arithmetization!

Plonk $\neq$ R1CS

A not basic at all example of arithmetization

The cost of each operation depends on the arithmetization!

Plonk $\neq$ R1CS

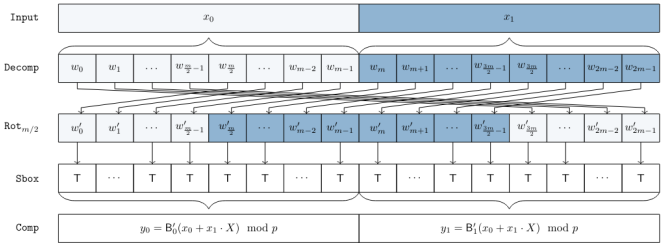


Figure 3: The Bar layer $\mathsf{B}' : \mathbb{F}_{p^n} \to \mathbb{F}_{p^n}$ for $n = 2$ in detail, including the decomposition, the rotation, the S-box, and the composition.

source: *Skyscraper: Fast Hashing on Big Primes*, `https://eprint.iacr.org/2025/058.pdf`

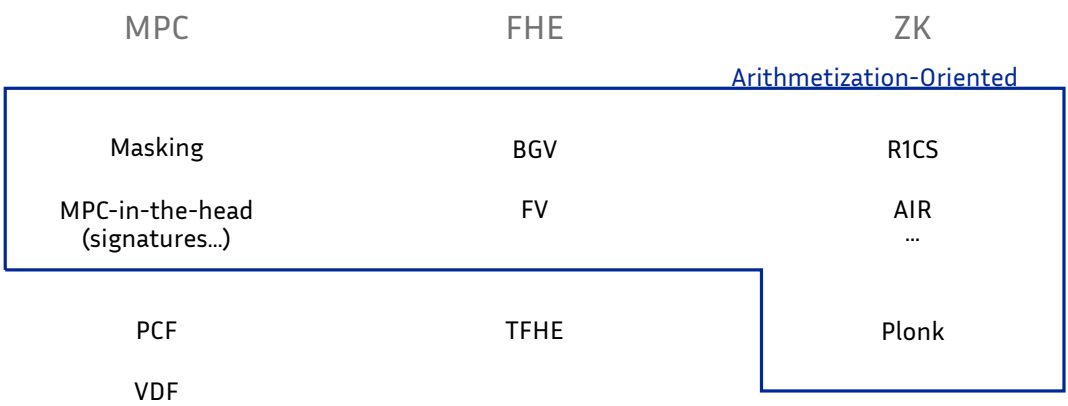
Symmetric Techniques for Advanced Protocols

MPC FHE ZK

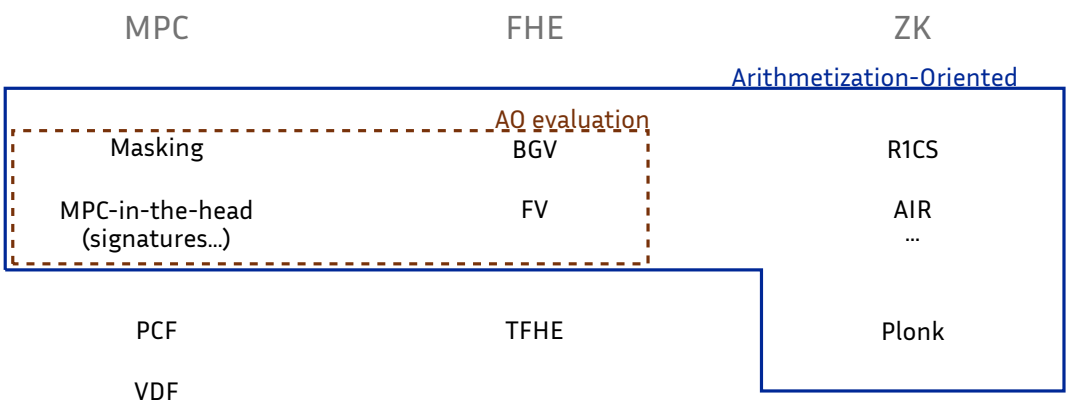
Symmetric Techniques for Advanced Protocols

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

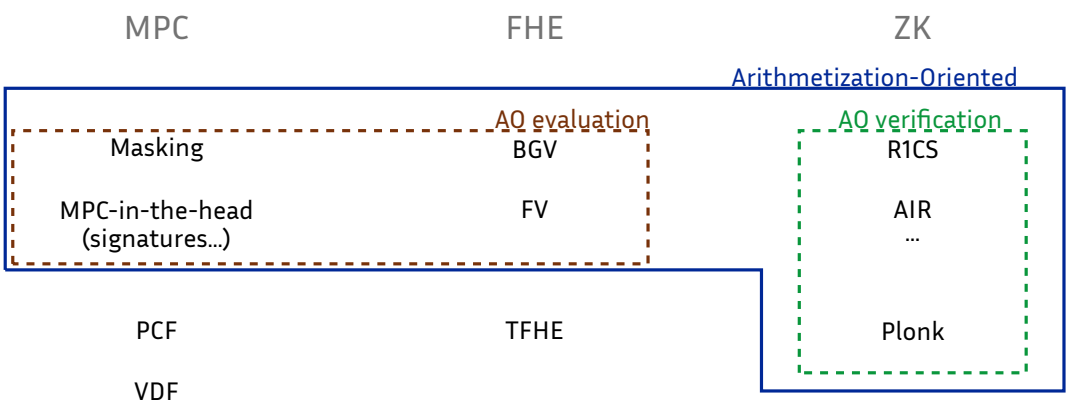
Symmetric Techniques for Advanced Protocols



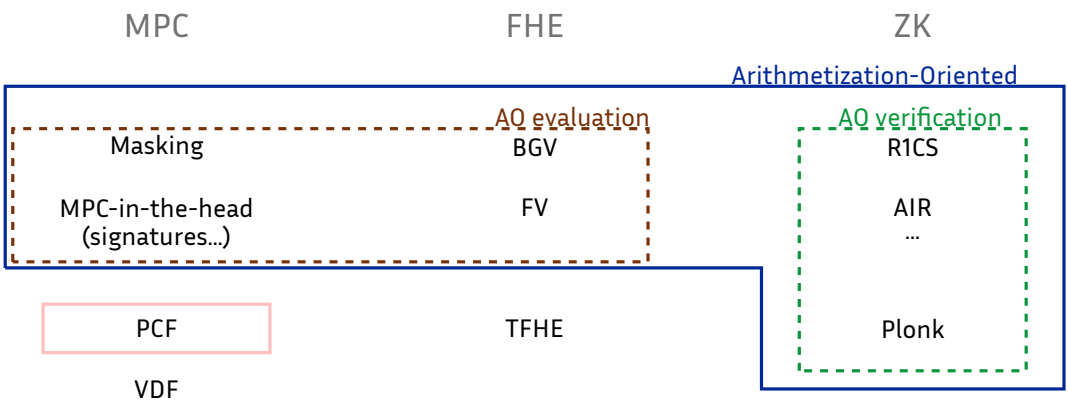
Symmetric Techniques for Advanced Protocols



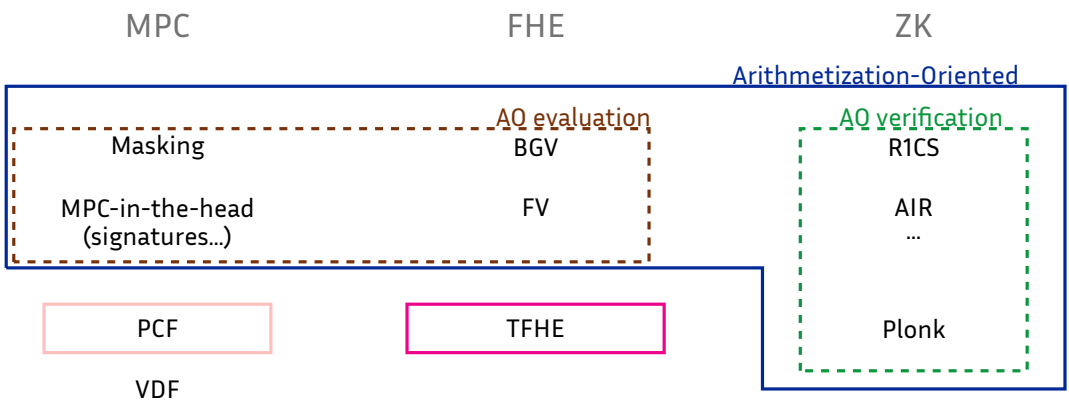
Symmetric Techniques for Advanced Protocols



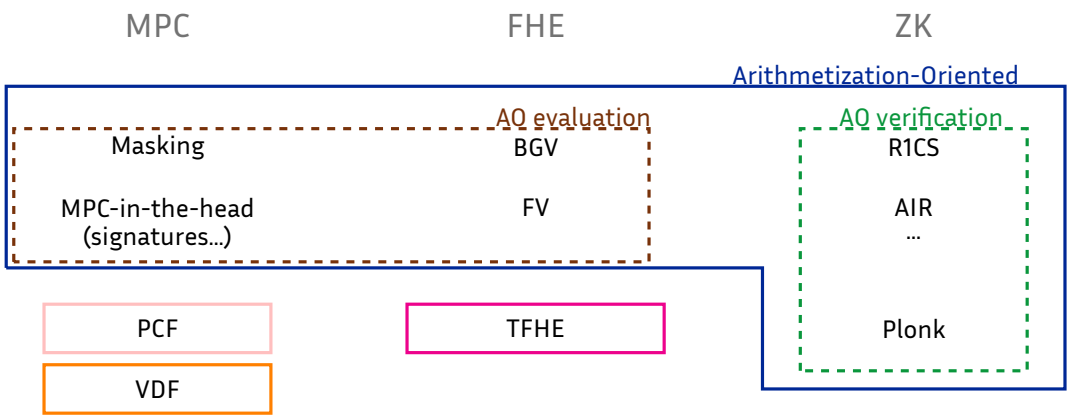
Symmetric Techniques for Advanced Protocols



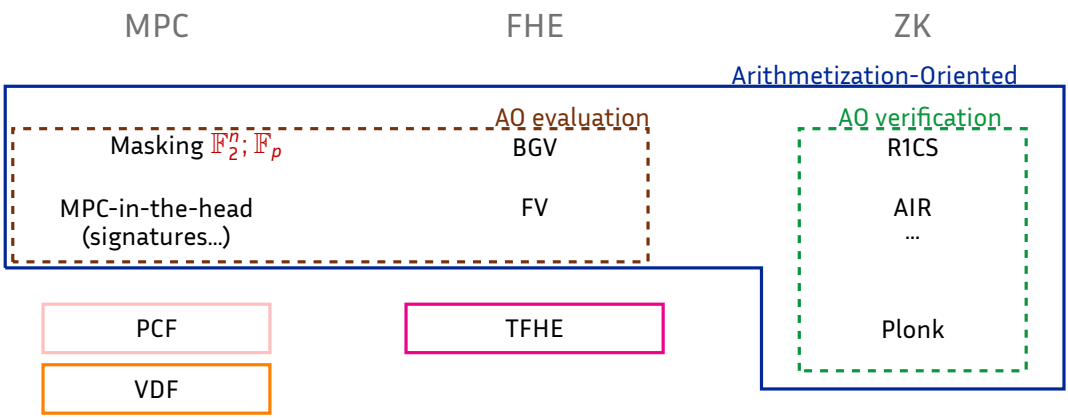
Symmetric Techniques for Advanced Protocols



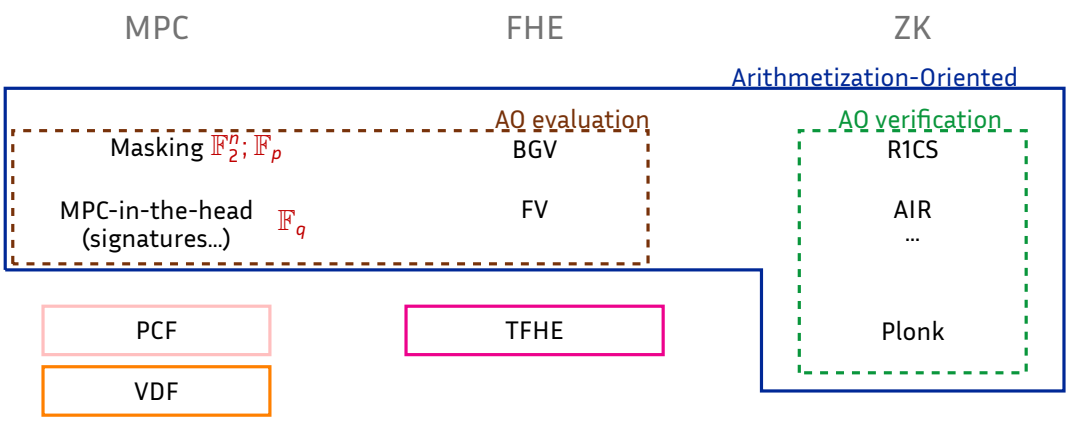
Symmetric Techniques for Advanced Protocols



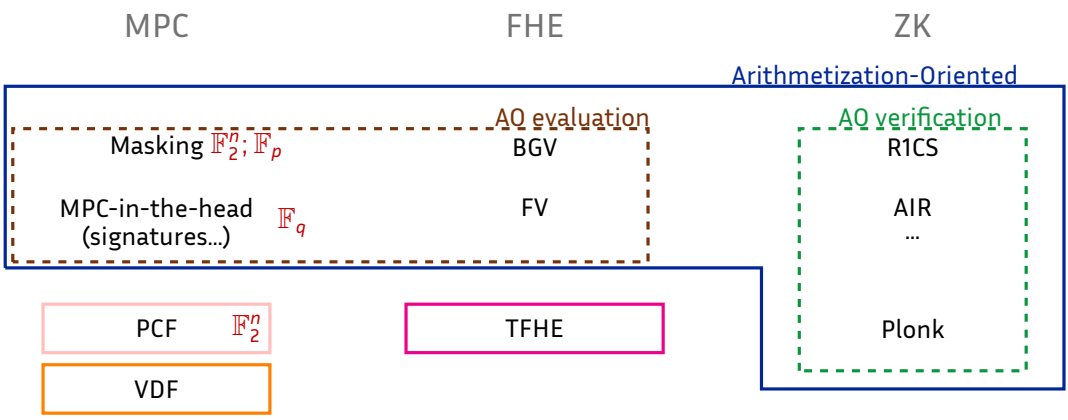
Symmetric Techniques for Advanced Protocols



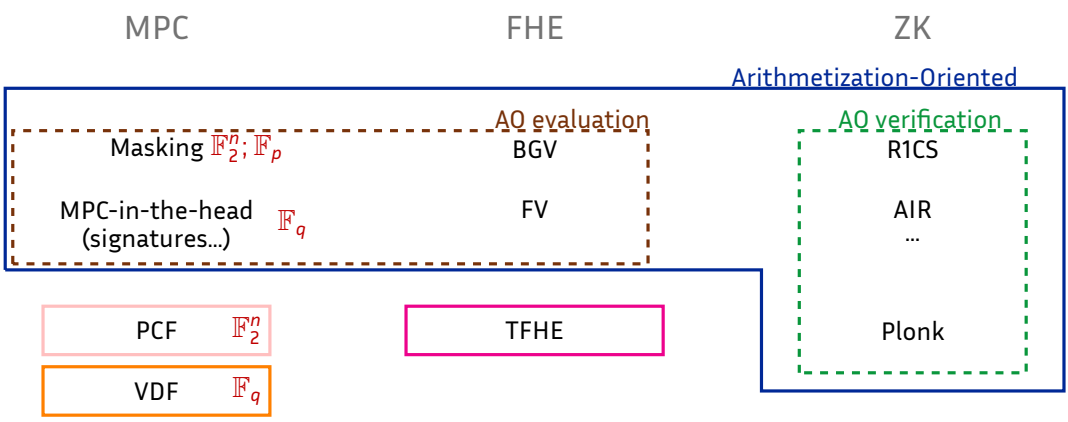
Symmetric Techniques for Advanced Protocols



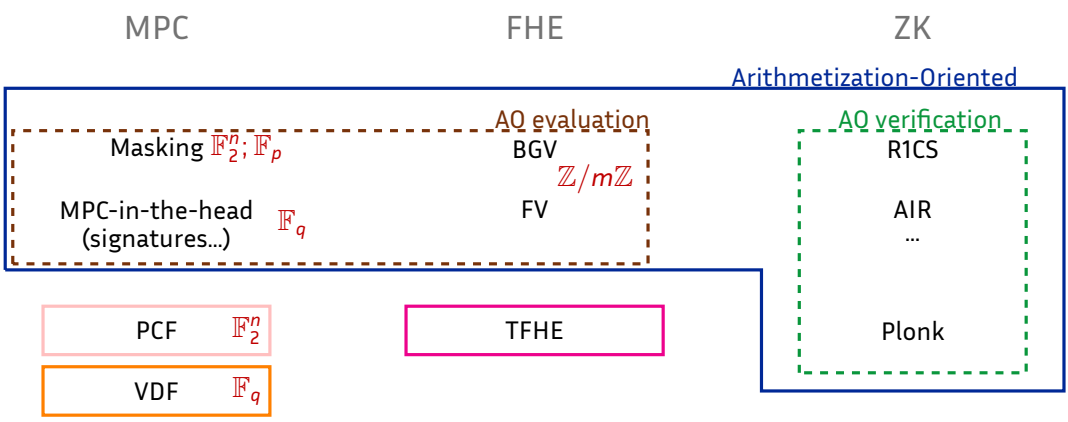
Symmetric Techniques for Advanced Protocols



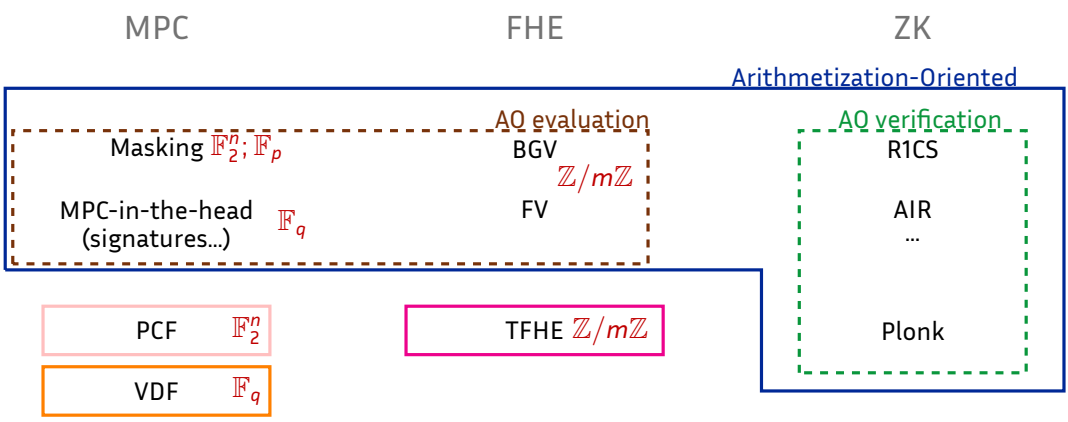
Symmetric Techniques for Advanced Protocols



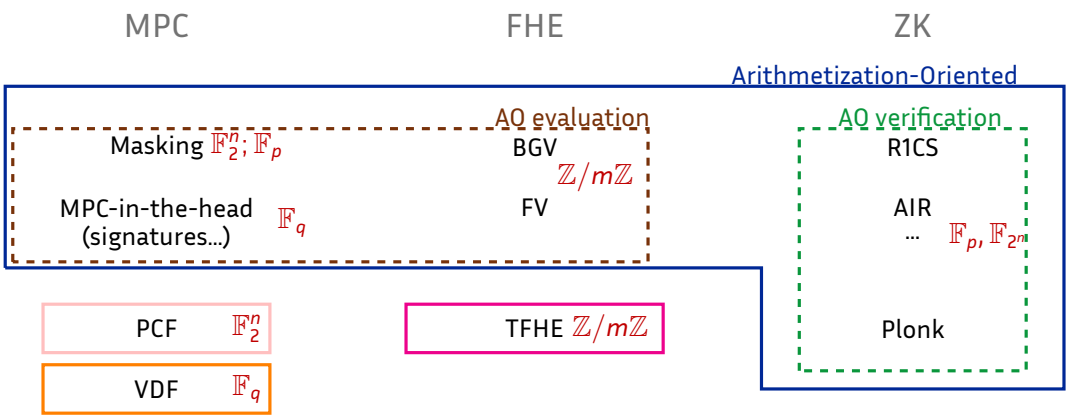
Symmetric Techniques for Advanced Protocols



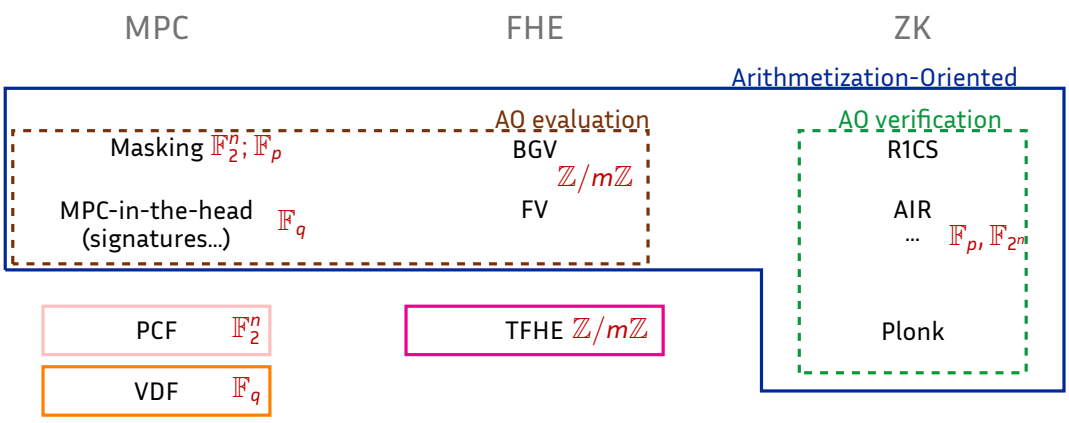
Symmetric Techniques for Advanced Protocols



Symmetric Techniques for Advanced Protocols



Symmetric Techniques for Advanced Protocols



It's kind of a mess!

Du passé, faisons table rase (?)

	Old World	New World

Du passé, faisons table rase (?)

	Old World	New World
Alphabet	$\mathbb{F}_2^n$	$\mathbb{Z}/q\mathbb{Z}$

Du passé, faisons table rase (?)

	Old World	New World
Alphabet	$\mathbb{F}_2^n$	$\mathbb{Z}/q\mathbb{Z}$
Operations	$\boxplus$, $\oplus$, AESenc, S-box...	$+$, x^3, PBS

Du passé, faisons table rase (?)

	Old World	**New World**
Alphabet	$\mathbb{F}_2^n$	$\mathbb{Z}/q\mathbb{Z}$
Operations	$\boxplus$, $\oplus$, `AESenc`, S-box...	$+$, x^3, PBS
Cost	# CPU instructions...	R1CS constraints, noise, # PBS...

Du passé, faisons table rase (?)

	Old World	**New World**
Alphabet	$\mathbb{F}_2^n$	$\mathbb{Z}/q\mathbb{Z}$
Operations	$\boxplus$, $\oplus$, AESenc, S-box...	$+$, x^3, PBS
Cost	# CPU instructions...	R1CS constraints, noise, # PBS...
Design strategy	Prevent diff./integral attacks $\rightarrow$ wide trail strategy	???

Plan of this Section

1 Full round attacks

2 These primitives are harder to design
- On designing a symmetric primitive
- New use cases: the need for arithmetizations
- STAP Examples

3 Causes of death

4 Conclusion

TFHE: corresponding stream ciphers

Elisabeth-4 $q = 2^4$

Uses a constant key register on which index-dependent non-linear functions are applied.

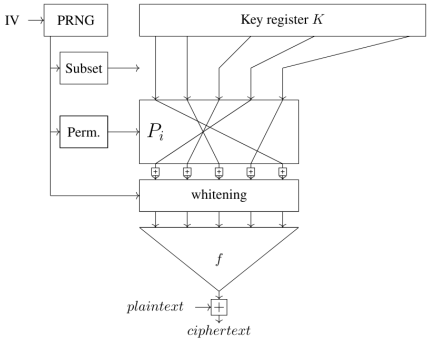


Fig. 1: The group filter permutator design

source: *Towards Case-Optimized Hybrid Homomorphic Encryption Featuring the Elisabeth Stream Cipher*

TFHE: corresponding stream ciphers

Elisabeth-4 $q = 2^4$

Uses a constant key register on which index-dependent non-linear functions are applied.

Gabriel... (Elisabeth-4 follow-ups)

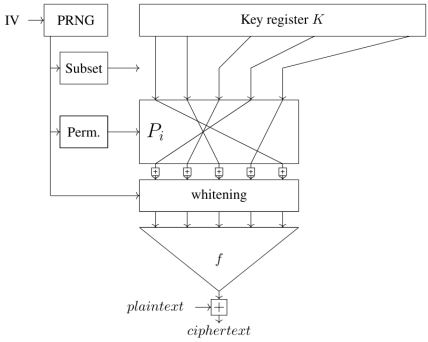


Fig. 1: The group filter permutator design

source: *Towards Case-Optimized Hybrid Homomorphic Encryption Featuring the Elisabeth Stream Cipher*

TFHE: corresponding stream ciphers

Elisabeth-4 $q = 2^4$
Uses a constant key register on which index-dependent non-linear functions are applied.

Gabriel... (Elisabeth-4 follow-ups)

FRAST $q = 2^4$ A block cipher in a CTR-mode variant.

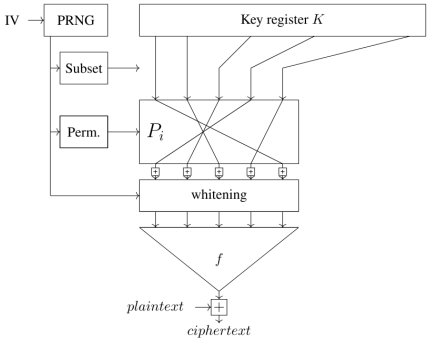


Fig. 1: The group filter permutator design

source: *Towards Case-Optimized Hybrid Homomorphic Encryption Featuring the Elisabeth Stream Cipher*

TFHE: corresponding stream ciphers

Elisabeth-4 $q = 2^4$
Uses a constant key register on which index-dependent non-linear functions are applied.

Gabriel... (Elisabeth-4 follow-ups)

FRAST $q = 2^4$ A block cipher in a CTR-mode variant.

Transistor $q = 2^4 + 1$
SNOW-like round structure

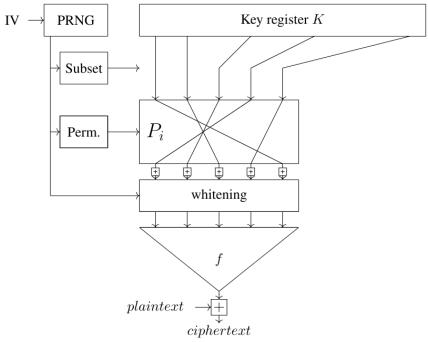


Fig. 1: The group filter permutator design

source: *Towards Case-Optimized Hybrid Homomorphic Encryption Featuring the Elisabeth Stream Cipher*

BGV/FV: corresponding stream ciphers

- ASTA $q = 2$ or large prime
  Use very few rounds with a low degree.
  Rely on large, randomly generated, nonce-dependent matrices.

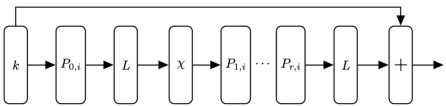


Figure 2: Generation of i-th block of DASTA.

source:
Dasta – Alternative Linear Layer for Rasta

BGV/FV: corresponding stream ciphers

·ASTA $q = 2$ or large prime

Use very few rounds with a low degree.

Rely on large, randomly generated, nonce-dependent matrices.

"Kreyvium" $q = 2$

Basically Trivium!

Binary state updated with NLFSRs.

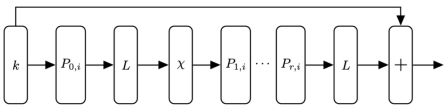


Figure 2: Generation of i-th block of DASTA.

source:

Dasta – Alternative Linear Layer for Rasta

BGV/FV: corresponding stream ciphers

·ASTA $q = 2$ or large prime

Use very few rounds with a low degree.

Rely on large, randomly generated, nonce-dependent matrices.

"Kreyvium" $q = 2$

Basically Trivium!

Binary state updated with NLFSRs.

HERA q large prime

A block cipher in a kind of CTR-mode variant.

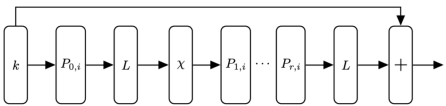


Figure 2: Generation of i-th block of DASTA.

source:

Dasta – Alternative Linear Layer for Rasta

ZK-Friendly Hash Functions

Poseidon no specific constraints q

SPN with partial layer of low degree monomial S-boxes.

Full rounds – partial round – full rounds.

ZK-Friendly Hash Functions

Poseidon no specific constraints q

SPN with partial layer of low degree monomial S-boxes.

Full rounds – partial round – full rounds.

Rescue q large prime

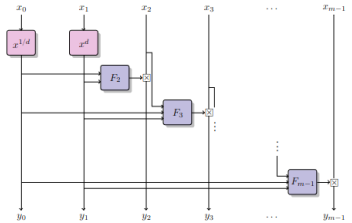
SPN with low degree monomials and their inverses.

ZK-Friendly Hash Functions

Poseidon no specific constraints q
SPN with partial layer of low degree monomial S-boxes.
Full rounds – partial round – full rounds.

Rescue q large prime
SPN with low degree monomials and their inverses.

Griffin q large prime
Feistel variant with multiplications instead.



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

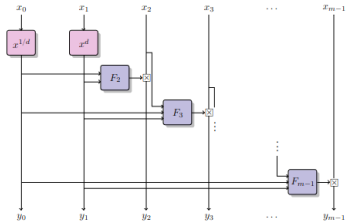
ZK-Friendly Hash Functions

Poseidon no specific constraints q
SPN with partial layer of low degree monomial S-boxes.
Full rounds – partial round – full rounds.

Rescue q large prime
SPN with low degree monomials and their inverses.

Griffin q large prime
Feistel variant with multiplications instead.

Anemoi $q = 2^n$ or large prime
Uses the "Flystel", a high degree S-box *CCZ-equivalent* to a function of low degree.



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

Plan of this Section

1 Full round attacks

2 These primitives are harder to design

3 Causes of death

4 Conclusion

Plan of this Section

1 Full round attacks

2 These primitives are harder to design

3 Causes of death
- A graveyard?
- PoSSo-Based Cryptanalysis
- The even-less-simple case

4 Conclusion

“Broken” Primitives

Use	Name	Security (intended)	Best Attack	
			Complexity	Type
TFHE	Elisabeth-4	128	88	Linearization
	FRAST	128	1*	Differential

“Broken” Primitives

Use	Name	Security (intended)	Best Attack	
			Complexity	Type
TFHE	Elisabeth-4	128	88	Linearization
	FRAST	128	1*	Differential
Plonk	Skysraper v1	128	20	Rebound

“Broken” Primitives

Use	Name	Security (intended)	Best Attack	
			Complexity	Type
TFHE	Elisabeth-4	128	88	Linearization
	FRAST	128	1*	Differential
Plonk	Skysraper v1	128	20	Rebound
ZK	Jarvis	128	83	ad hoc PoSSo
	Anemoi	128	80	Resultants
	Arion	128	90	
	Griffin	128	55	
	Rescue	512	475	

“Broken” Primitives

Use		ck Type
TFHE		nearization ifferential
Plonk	S	Rebound
ZK		hoc PoSSo esultants

An appetizer: The case of FRAST

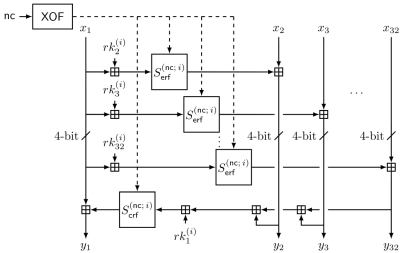


Figure 2: The i-th round function of FRAST. $rk_1^{(i)}, \ldots, rk_{32}^{(i)}$ are the i-th round keys.

An appetizer: The case of FRAST

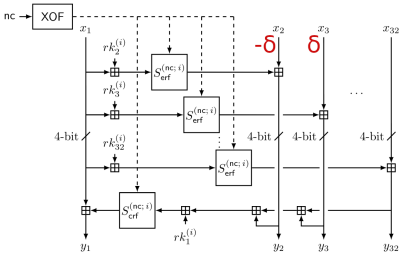


Figure 2: The i-th round function of FRAST. $rk_1^{(i)}, \ldots, rk_{32}^{(i)}$ are the i-th round keys.

1 Proba 1 iterated differential: $(0, -\delta, \delta, 0, \ldots, 0)$

An appetizer: The case of FRAST

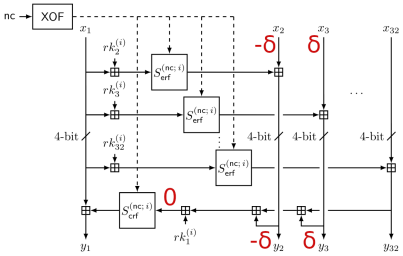


Figure 2: The i-th round function of FRAST. $rk_1^{(i)}, \ldots, rk_{32}^{(i)}$ are the i-th round keys.

1. Proba 1 iterated differential: $(0, -\delta, \delta, 0, \ldots, 0)$
2. Proba 1 iterated differential: $(0, 0, \delta, 0, \ldots, 0, -\delta)$

An appetizer: The case of FRAST

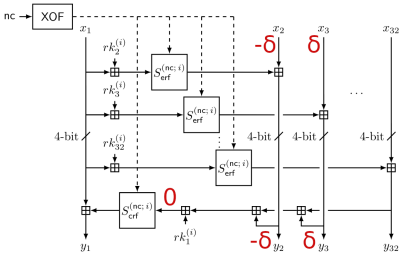


Figure 2: The i-th round function of FRAST. $rk_1^{(i)}, \ldots, rk_{32}^{(i)}$ are the i-th round keys.

1. Proba 1 iterated differential: $(0, -\delta, \delta, 0, \ldots, 0)$
2. Proba 1 iterated differential: $(0, 0, \delta, 0, \ldots, 0, -\delta)$
3. Proba 1 iterated differential: $(0, \delta_2, \delta_3, \ldots, 0, \delta_{32})$ provided that $\sum_i \delta_i = 0$

An appetizer: The case of FRAST

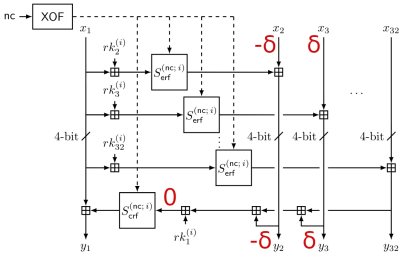


Figure 2: The i-th round function of FRAST. $rk_1^{(i)}, \ldots, rk_{32}^{(i)}$ are the i-th round keys.

1. Proba 1 iterated differential: $(0, -\delta, \delta, 0, \ldots, 0)$
2. Proba 1 iterated differential: $(0, 0, \delta, 0, \ldots, 0, -\delta)$
3. Proba 1 iterated differential: $(0, \delta_2, \delta_3, \ldots, 0, \delta_{32})$ provided that $\sum_i \delta_i = 0$

Almost Affinity

Let E_K be the internal block cipher of FRAST

$$\Pr_{\Delta}\left[\forall x, E_K(x + \Delta) = E_K(x) + \Delta\right] = 2^{-8}$$

Plan of this Section

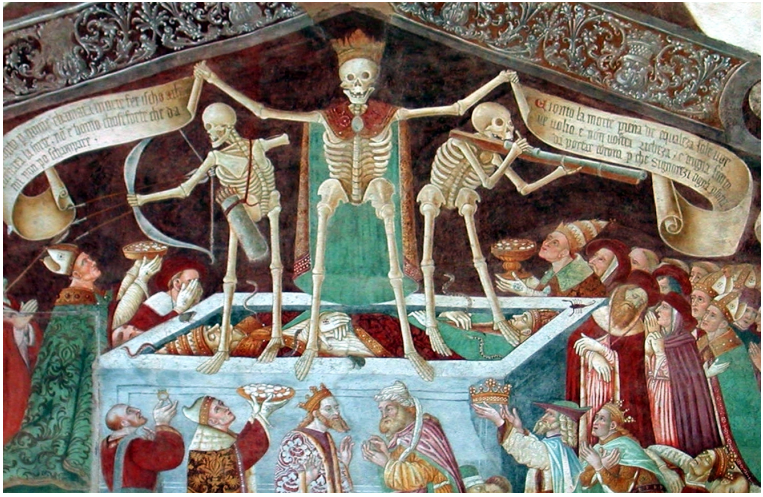
1 Full round attacks

2 These primitives are harder to design

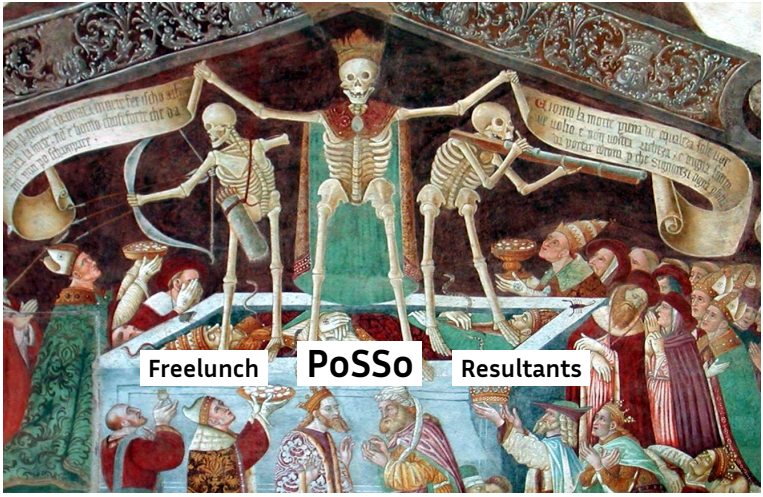
3 Causes of death
- A graveyard?
- PoSSo-Based Cryptanalysis
- The even-less-simple case

4 Conclusion

Allegory



Allegory



Definition of PoSSo

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 of PoSSo

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 of PoSSo

Polynomial System Solving

$$\begin{cases} p_1(x_1, \ldots, x_N) = 0 \\ \vdots \\ p_{k-1}(x_1, \ldots, x_N) = 0 \\ p_k(x_1, \ldots, x_N) = 0 \end{cases} \longrightarrow \begin{cases} x_1 = c_1 \\ \vdots \\ x_{N-1} = c_{N-1} \\ x_N = c_N \end{cases}$$

PoSSo magic!

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

Steps of a PoSSo-Based Attack

Write a system of equations.

Steps of a PoSSo-Based Attack

Write a system of equations.

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

Steps of a PoSSo-Based Attack

Write a system of equations.

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

Steps of a PoSSo-Based 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)

Steps of a PoSSo-Based 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)

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

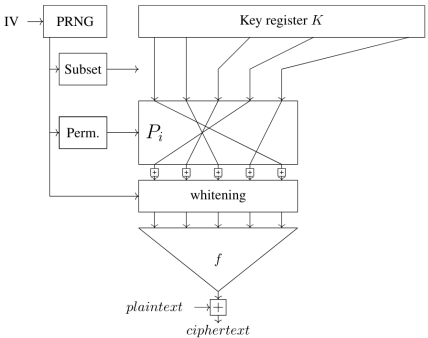


Fig. 1: The group filter permutator design

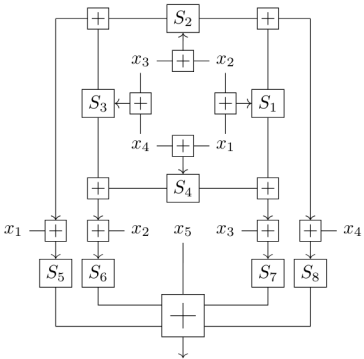


Fig. 2: Elisabeth-4 's 5 to 1 inner function.

The particular case of Elisabeth (2/2)

Basic Linearization

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

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

The particular case of Elisabeth (2/2)

Basic Linearization

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

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!

The particular case of Elisabeth (2/2)

Basic Linearization

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

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

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

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}

The simplest case: linearization

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.

The simplest case: linearization

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)

The simplest case: linearization

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)

The best algorithm was not the one considered, and its complexity scaled differently.

Plan of this Section

1 Full round attacks

2 These primitives are harder to design

3 Causes of death
- A graveyard?
- PoSSo-Based Cryptanalysis
- The even-less-simple case

4 Conclusion

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

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

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**

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

$$\begin{cases} p_1(x_1,\ldots,x_N)=0 \\ \vdots \\ p_{k-1}(x_1,\ldots,x_N)=0 \\ p_k(x_1,\ldots,x_N)=0 \end{cases} \quad \begin{cases} g_1(x_1,\ldots,x_N)=0 \\ \vdots \\ g_{\kappa-1}(x_1,\ldots,x_N)=0 \\ g_\kappa(x_1,\ldots,x_N)=0 \end{cases} \quad \begin{cases} g^*_1(x_1,\ldots,x_N)=0 \\ \vdots \\ g^*_{N-1}(x_{N-1},x_N)=0 \\ g^*_N(x_N)=0 \end{cases} \quad g^*_N(x_N)=0 \to x_N$$

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** 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

How to prevent it?

Usual goal

Ensure that the attack has a time complexity beyond 2^{128}.

How to prevent it?

Usual goal

Ensure that the attack has a time complexity beyond 2^{128}.

1. Experimentally, **F4/F5** is by far the slowest
2. Time complexity (m is the number of equations):

$$O\left(\binom{m-1+D_{\mathrm{reg}}}{m-1}^{\omega}\right)$$

How to prevent it?

Usual goal

Ensure that the attack has a time complexity beyond 2^{128}.

1. Experimentally, **F4/F5** is by far the slowest
2. Time complexity (m is the number of equations):

$$O\left(\binom{m-1+D_{\mathrm{reg}}}{m-1}^{\omega}\right)$$

3. Estimate D_{reg}, plug in the numbers... Voilà!

How to prevent it?

Usual goal

Ensure that the attack has a time complexity beyond 2^{128}.

1. Experimentally, **F4/F5** is by far the slowest
2. Time complexity (m is the number of equations):

$$O\left(\binom{m-1+D_{\mathrm{reg}}}{m-1}^{\omega}\right)$$

3. Estimate D_{reg}, plug in the numbers... Voilà!

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

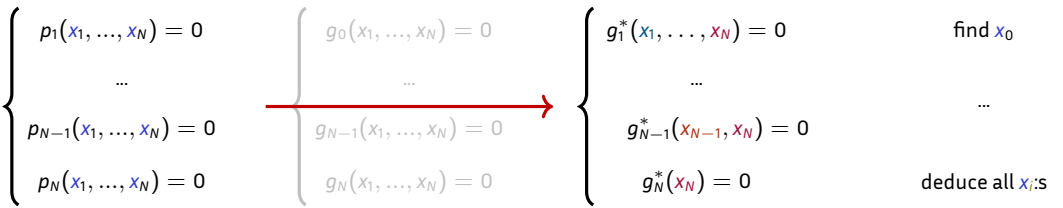
find x_0

...

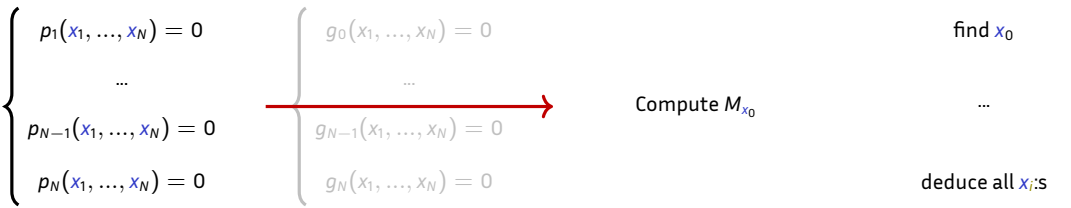
deduce all x_i:s

4. Final resolution

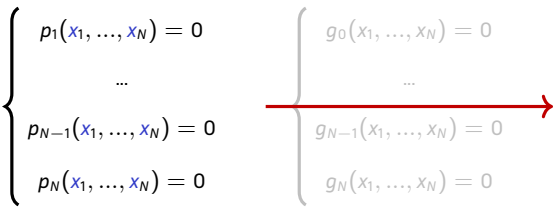
A Better Solving Strategy (Sometimes): the Freelunch



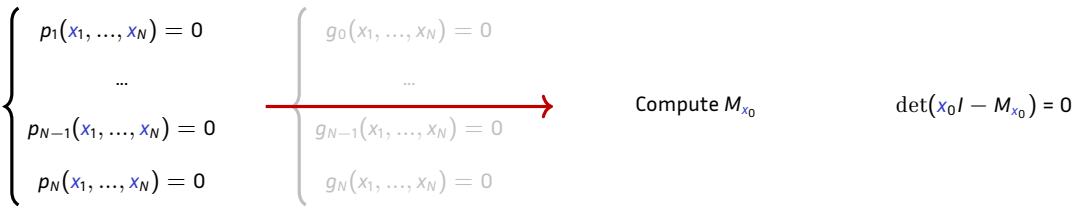
A Better Solving Strategy (Sometimes): the Freelunch



A Better Solving Strategy (Sometimes): the Freelunch



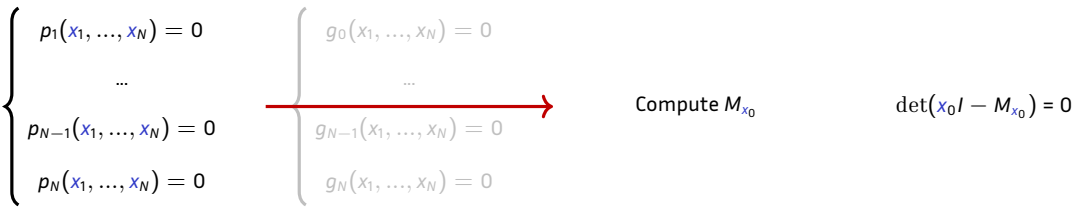
A Better Solving Strategy (Sometimes): the Freelunch



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



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

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

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

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

Towards better security arguments

Lessons Learnt

1. The best solving algorithm might not be the one we think of at first

Towards better security arguments

Lessons Learnt

1. The best solving algorithm might not be the one we think of at first
2. Dedicated solving algorithm will be even better

Towards better security arguments

Lessons Learnt

1. The best solving algorithm might not be the one we think of at first
2. Dedicated solving algorithm will be even better
3. The **system** to solve might not be the one we think of at first either

Towards better security arguments

Lessons Learnt

1. The best solving algorithm might not be the one we think of at first
2. Dedicated solving algorithm will be even better
3. The **system** to solve might not be the one we think of at first either

All is not lost: univariate root finding is always there, so D_I is[a] a good metric!

[a] Hopefully...

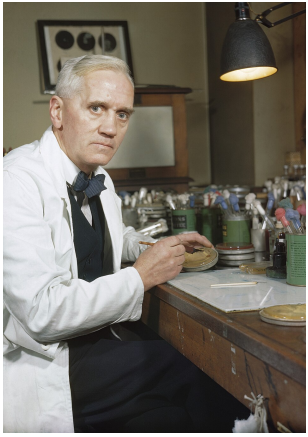
Towards better security arguments

Lessons Learnt

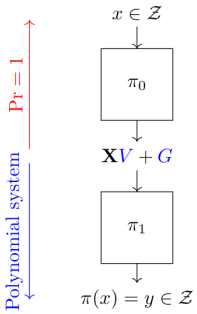
1. The best solving algorithm might not be the one we think of at first
2. Dedicated solving algorithm will be even better
3. The **system** to solve might not be the one we think of at first either

All is not lost: univariate root finding is always there, so D_I is[a] a good metric!

[a] Hopefully...



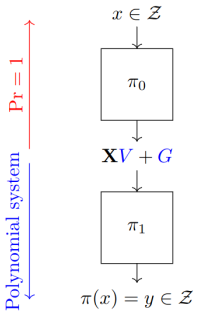
Round Skips: a limit of D_I...



Principle

Use the degrees of freedom to add equations that decrease the ideal degree.

Round Skips: a limit of D_I...

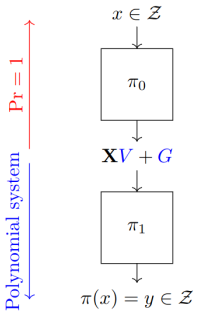


Principle

Use the degrees of freedom to add equations that decrease the ideal degree.

- At first, found using “subspace” tricks.
- Better techniques exist now:
  Skipping Class: Algebraic Attacks exploiting weak matrices and operation modes of Poseidon2(b), Merz and Rodgriguez Garcia,
  `https://eprint.iacr.org/2026/306`

Round Skips: a limit of D_I...



Principle

Use the degrees of freedom to add equations that decrease the ideal degree.

- At first, found using “subspace” tricks.
- Better techniques exist now:
  Skipping Class: Algebraic Attacks exploiting weak matrices and operation modes of Poseidon2(b), Merz and Rodgriguez Garcia,
  `https://eprint.iacr.org/2026/306`

Bigger blocks $\longrightarrow$ higher D_I, but also more degrees freedom!

Plan of this Section

1 Full round attacks

2 These primitives are harder to design

3 Causes of death

4 Conclusion

Plan of this Section

1 Full round attacks

2 These primitives are harder to design

3 Causes of death

4 Conclusion
- Is it *this* bad?
- Where to go from here?

The particular case of ZK-friendly HF

Uncertainty Different protocols use different primes $\implies$ different possible values for α.

The particular case of ZK-friendly HF

Uncertainty Different protocols use different primes $\Longrightarrow$ different possible values for α.

Flexibility These algorithms give the user (and the attacker!) a lot of freedom

Table 1: *Number of Rounds of* `Anemoi`.

α	3	5	7	11
$\ell = 1$	21	21	20	19
$\ell = 2$	14	14	13	13
$\ell = 3$	12	12	12	11
$\ell = 4$	12	12	11	11
$\ell = 6$	10	10	10	10
$\ell = 8$	10	10	9	9

(a) *When* $s = 128$.

α	3	5	7	11
$\ell = 1$	37	37	36	35
$\ell = 2$	22	22	21	21
$\ell = 3$	17	17	17	17
$\ell = 4$	16	16	15	15
$\ell = 6$	13	13	13	13
$\ell = 8$	12	12	11	11

(b) *When* $s = 256$.

The particular case of ZK-friendly HF

Uncertainty Different protocols use different primes $\implies$ different possible values for α.

Flexibility These algorithms give the user (and the attacker!) a lot of freedom

Table 1: *Number of Rounds of* `Anemoi`.

α	3	5	7	11
$\ell = 1$	21	21	20	19
$\ell = 2$	14	14	13	13
$\ell = 3$	12	12	12	11
$\ell = 4$	12	12	11	11
$\ell = 6$	10	10	10	10
$\ell = 8$	10	10	9	9

(a) *When* $s = 128$.

α	3	5	7	11
$\ell = 1$	37	37	36	35
$\ell = 2$	22	22	21	21
$\ell = 3$	17	17	17	17
$\ell = 4$	16	16	15	15
$\ell = 6$	13	13	13	13
$\ell = 8$	12	12	11	11

(b) *When* $s = 256$.

Comparison $\neq$ Security evaluation Cryptanalysts focus on **comparing instances**

The particular case of ZK-friendly HF

Uncertainty Different protocols use different primes $\implies$ different possible values for α.

Flexibility These algorithms give the user (and the attacker!) a lot of freedom

Table 1: *Number of Rounds of* `Anemoi`.

α	3	5	7	11
$\ell = 1$	21	21	20	19
$\ell = 2$	14	14	13	13
$\ell = 3$	12	12	12	11
$\ell = 4$	12	12	11	11
$\ell = 6$	10	10	10	10
$\ell = 8$	10	10	9	9

(a) *When* $s = 128$.

α	3	5	7	11
$\ell = 1$	37	37	36	35
$\ell = 2$	22	22	21	21
$\ell = 3$	17	17	17	17
$\ell = 4$	16	16	15	15
$\ell = 6$	13	13	13	13
$\ell = 8$	12	12	11	11

(b) *When* $s = 256$.

Comparison $\neq$ Security evaluation Cryptanalysts focus on **comparing instances**

Is there a set of parameters for which my attack works?

$\neq$

What is the security of the variant I care about?

Looking at Rescue

This slide and the following ones are heavily based on

Improved Resultant Attack against Arithmetization-Oriented Primitives, Bariant et al.
`https://eprint.iacr.org/2025/259`

Looking at Rescue

This slide and the following ones are heavily based on

Improved Resultant Attack against Arithmetization-Oriented Primitives, Bariant et al.
`https://eprint.iacr.org/2025/259`

r	`iteRes`	`uniSol`	[26]	Designers' estimation [23]
6	49	40	60	-
7	57	45	69	-
8	64	50	81	-
12	93	70	-	80
18	137	99	-	128
35	257	181	-	256
66	475	330	-	512

Table 3. Theoretical complexities (in $\log_2$ $\mathbb{F}_q$-multiplications) for attacking variants of Rescue with $t = \alpha = 3$ and $\log_2(q) \approx 512$. Our analysis show that the complexity of `sysGen` is negligible compared to the rest of the attack.

Looking at Rescue

This slide and the following ones are heavily based on

Improved Resultant Attack against Arithmetization-Oriented Primitives, Bariant et al.
`https://eprint.iacr.org/2025/259`

r	`iteRes`	`uniSol`	[26]	Designers' estimation [23]
6	49	40	60	-
7	57	45	69	-
8	64	50	81	-
12	93	70	-	80
18	137	99	-	128
35	257	181	-	256
66	475	330	-	512

Table 3. Theoretical complexities (in $\log_2 \mathbb{F}_q$-multiplications) for attacking variants of Rescue with $t = \alpha = 3$ and $\log_2(q) \approx 512$. Our analysis show that the complexity of `sysGen` is negligible compared to the rest of the attack.

All instance are fine except for a single one (the 512-bit one)

Looking at Anemoi

α	ℓ	r	sysGen	iteRes [3, polyDet] [26]		α	ℓ	r	sysGen	iteRes [3, polyDet] [26]	
3	1	21	78	80	118	110	3	1	37	129	130	203	191
	2	14	89	89	-	-		2	22	133	134	-	-
	3	12	105	104	-	-		3	17	144	144	-	-
	4	12	131	131	-	-		4	16	172	172	-	-
	6	10	154	153	-	-		6	13	199	198	-	-
5	1	21	91	96	156	133	5	1	37	151	157	270	231
	2	14	108	113	-	-		2	22	164	169	-	-
	3	12	131	135	-	-		3	17	182	186	-	-
	4	12	167	171	-	-		4	16	221	224	-	-
	6	10	199	202	-	-		6	13	258	262	-	-
7	1	20	96	103	174	141	7	1	36	162	170	307	252
	2	13	113	119	-	-		2	21	177	183	-	-
	3	12	148	153	-	-		3	17	206	212	-	-
	4	11	174	179	-	-		4	15	236	241	-	-
	6	10	227	232	-	-		6	13	296	300	-	-
11	1	19	102	111	198	158	11	1	35	179	187	358	288
	2	13	129	137	-	-		2	21	203	211	-	-
	3	11	156	163	-	-		3	17	238	245	-	-
	4	11	202	208	-	-		4	15	274	280	-	-
	6	10	264	270	-	-		6	13	345	351	-	-

Table 4. Theoretical complexities (in $\log_2 \mathbb{F}_q$-multiplications) for attacking full-round variants of Anemoi in odd characteristic proposed for 128-bit security on the left-hand side table, and 256-bit security on the right-hand side table. To stay consistent with the assumptions of previous work, we used $\omega = 2.81$ for the attack of [3] and $\omega = 2.376$ for [26]. Experiments show that the attack on odd-characteristic Anemoi is heavily dominated by `iteRes` in practice.

Looking at Anemoi

α	ℓ	r	sysGen	iteRes	[3, polyDet]	[26]
3	1	21	78	80	118	110
	2	14	89	89	-	-
	3	12	105	104	-	-
	4	12	131	131	-	-
	6	10	154	153	-	-
5	1	21	91	96	156	133
	2	14	108	113	-	-
	3	12	131	135	-	-
	4	12	167	171	-	-
	6	10	199	202	-	-
7	1	20	96	103	174	141
	2	13	113	119	-	-
	3	12	148	153	-	-
	4	11	174	179	-	-
	6	10	227	232	-	-
11	1	19	102	111	198	158
	2	13	129	137	-	-
	3	11	156	163	-	-
	4	11	202	208	-	-
	6	10	264	270	-	-

α	ℓ	r	sysGen	iteRes	[3, polyDet]	[26]
3	1	37	129	130	203	191
	2	22	133	134	-	-
	3	17	144	144	-	-
	4	16	172	172	-	-
	6	13	199	198	-	-
5	1	37	151	157	270	231
	2	22	164	169	-	-
	3	17	182	186	-	-
	4	16	221	224	-	-
	6	13	258	262	-	-
7	1	36	162	170	307	252
	2	21	177	183	-	-
	3	17	206	212	-	-
	4	15	236	241	-	-
	6	13	296	300	-	-
11	1	35	179	187	358	288
	2	21	203	211	-	-
	3	17	238	245	-	-
	4	15	274	280	-	-
	6	13	345	351	-	-

Table 4. Theoretical complexities (in $\log_2 \mathbb{F}_q$-multiplications) for attacking full-round variants of Anemoi in odd characteristic proposed for 128-bit security on the left-hand side table, and 256-bit security on the right-hand side table. To stay consistent with the assumptions of previous work, we used $\omega = 2.81$ for the attack of [3] and $\omega = 2.376$ for [26]. Experiments show that the attack on odd-characteristic Anemoi is heavily dominated by `iteRes` in practice.

Looking at Anemoi

α	ℓ	r	sysGen	iteRes	[3, polyDet]	[26]
3	1	21	78	80	118	110
	2	14	89	89	-	-
	3	12	105	104	-	-
	4	12	131	131	-	-
	6	10	154	153	-	-
5	1	21	91	96	156	133
	2	14	108	113	-	-
	3	12	131	135	-	-
	4	12	167	171	-	-
	6	10	199	202	-	-
7	1	20	96	103	174	141
	2	13	113	119	-	-
	3	12	148	153	-	-
	4	11	174	179	-	-
	6	10	227	232	-	-
11	1	19	102	111	198	158
	2	13	129	137	-	-
	3	11	156	163	-	-
	4	11	202	208	-	-
	6	10	264	270	-	-

α	ℓ	r	sysGen	iteRes	[3, polyDet]	[26]
3	1	37	129	130	203	191
	2	22	133	134	-	-
	3	17	144	144	-	-
	4	16	172	172	-	-
	6	13	199	198	-	-
5	1	37	151	157	270	231
	2	22	164	169	-	-
	3	17	182	186	-	-
	4	16	221	224	-	-
	6	13	258	262	-	-
7	1	36	162	170	307	252
	2	21	177	183	-	-
	3	17	206	212	-	-
	4	15	236	241	-	-
	6	13	296	300	-	-
11	1	35	179	187	358	288
	2	21	203	211	-	-
	3	17	238	245	-	-
	4	15	274	280	-	-
	6	13	345	351	-	-

Table 4. Theoretical complexities (in $\log_2 \mathbb{F}_q$-multiplications) for attacking full-round variants of Anemoi in odd characteristic proposed for 128-bit security on the left-hand side table, and 256-bit security on the right-hand side table. To stay consistent with the assumptions of previous work, we used $\omega = 2.81$ for the attack of [3] and $\omega = 2.376$ for [26]. Experiments show that the attack on odd-characteristic Anemoi is heavily dominated by `iteRes` in practice.

- Bigger blocks $\rightarrow$ more secure
- Bigger α $\rightarrow$ more secure

Looking at Anemoi

α	ℓ	r	sysGen	iteRes	[3, polyDet]	[26]
3	1	21	78	80	118	110
	2	14	89	89	-	-
	3	12	105	104	-	-
	4	12	131	131	-	-
	6	10	154	153	-	-
5	1	21	91	96	156	133
	2	14	108	113	-	-
	3	12	131	135	-	-
	4	12	167	171	-	-
	6	10	199	202	-	-
7	1	20	96	103	174	141
	2	13	113	119	-	-
	3	12	148	153	-	-
	4	11	174	179	-	-
	6	10	227	232	-	-
11	1	19	102	111	198	158
	2	13	129	137	-	-
	3	11	156	163	-	-
	4	11	202	208	-	-
	6	10	264	270	-	-

α	ℓ	r	sysGen	iteRes	[3, polyDet]	[26]
3	1	37	129	130	203	191
	2	22	133	134	-	-
	3	17	144	144	-	-
	4	16	172	172	-	-
	6	13	199	198	-	-
5	1	37	151	157	270	231
	2	22	164	169	-	-
	3	17	182	186	-	-
	4	16	221	224	-	-
	6	13	258	262	-	-
7	1	36	162	170	307	252
	2	21	177	183	-	-
	3	17	206	212	-	-
	4	15	236	241	-	-
	6	13	296	300	-	-
11	1	35	179	187	358	288
	2	21	203	211	-	-
	3	17	238	245	-	-
	4	15	274	280	-	-
	6	13	345	351	-	-

Table 4. Theoretical complexities (in $\log_2 \mathbb{F}_q$-multiplications) for attacking full-round variants of Anemoi in odd characteristic proposed for 128-bit security on the left-hand side table, and 256-bit security on the right-hand side table. To stay consistent with the assumptions of previous work, we used $\omega = 2.81$ for the attack of [3] and $\omega = 2.376$ for [26]. Experiments show that the attack on odd-characteristic Anemoi is heavily dominated by `iteRes` in practice.

- Bigger blocks $\longrightarrow$ more secure
- Bigger α $\longrightarrow$ more secure

When these parameters increase, other attacks take over (in the initial security analysis)

Plan of this Section

1 Full round attacks

2 These primitives are harder to design

3 Causes of death

4 Conclusion
- Is it *this* bad?
- Where to go from here?

They break because we are still learning

Glaring oversights aside, the main problem was
a poor understanding of relevant attack techniques.

They break because we are still learning

Glaring oversights aside, the main problem was
a poor understanding of relevant attack techniques.

Too many designs, too little cryptanalysis!

They break because we are still learning

Glaring oversights aside, the main problem was
a poor understanding of relevant attack techniques.

Too many designs, too little cryptanalysis!

Primitive factory vs. primitive

We still need to figure out a good way to specify/secure “factories”.

They break because we are still learning

Glaring oversights aside, the main problem was
a poor understanding of relevant attack techniques.

Too many designs, too little cryptanalysis!

Primitive factory vs. primitive

We still need to figure out a good way to specify/secure “factories”. (or do we?)

Potential directions for hardening

Ideal Degree

The “boring/fastest” step of PoSSo is the only with a reliable complexity.

$\Longrightarrow$ security arguments based on D_I are the future!

Potential directions for hardening

Ideal Degree

The “boring/fastest” step of PoSSo is the only with a reliable complexity.

$\Longrightarrow$ security arguments based on D_I are the future!

$$D_I \text{ vs. } D_I^2?$$

Potential directions for hardening

Ideal Degree

The “boring/fastest” step of PoSSo is the only with a reliable complexity.

$\Longrightarrow$ security arguments based on D_I are the future!

$$D_I \text{ vs. } D_I^2?$$

Easy security increase?

Like in public key, express attack efficiency in terms of a single quantity (number of rounds?) that is easy to increase if necessary

Potential directions for hardening

Ideal Degree

The “boring/fastest” step of PoSSo is the only with a reliable complexity.

$\Longrightarrow$ security arguments based on D_I are the future!

$$D_I \text{ vs. } D_I^2?$$

Easy security increase?

Like in public key, express attack efficiency in terms of a single quantity (number of rounds?) that is easy to increase if necessary

Thank You!

Looking at Arion

e	t	r	sysGen	iteRes	uniSol	[3, polyDet]
3	3	6	75	90	73	128
	4	6	82	96	79	134
	5	5	73	87	71	114
	6	5	78	91	75	119
	8	4	69	82	67	98
5	3	6	79	94	76	132
	4	5	72	86	70	113
	5	5	76	90	74	118
	6	5	81	95	78	122
	8	4	71	85	70	101

e	t	r	sysGen	iteRes	uniSol	[3, polyDet]
3	3	5	63	78	62	104
	4	4	55	69	54	84
	5	4	59	73	58	88
	6	4	62	76	61	92
	8	4	69	82	67	98
5	3	4	53	68	53	83
	4	4	57	72	57	87
	5	4	61	75	60	91
	6	4	65	79	64	94
	8	4	71	85	70	101

Table 2. Theoretical complexities (in $\log_2$ $\mathbb{F}_q$-multiplications) of attacks against full-round variants of Arion with 128-bit security using $\alpha = 121$. Normal parameters are in the left table, and the more aggressive α-Arion parameters are in the right table.

Looking at Griffin

α	t	r	`sysGen`	`iteRes`	`uniSol`	[3, `polyDet`]
3	3	16	92	92	83	120
	4	15	87	87	78	112
	8	11	63	61	57	76
	$\geq$12	10	55	53	51	64
5	3	14	102	106	92	141
	4	11	81	85	75	110
	8	9	63	66	58	81
	$\geq$12	9	60	62	56	74

Table 1. Complexity (in $\log_2$ $\mathbb{F}_q$-multiplications) of the steps of our attack on full-round variants of Griffin with 128-bit security claim.