Arithmetization-Oriented Symmetric Cryptography: Why and How?

Léo Perrin[1]

including joint works with

Clémence Bouvier, Pierre Briaud, Pyrros Chaidos, Robin Salen, Vesselin Velichkov, and Danny Willems,

Inria, Paris

20th of October 2022

informatics mathematics

Conclusion

A whole new world is opening in symmetric cryptography, that is more "algebraic" and where $\approx$ everything remains to be done.

Outline

Plan of this Section

Plan of this Section

Hash Functions

In what follows, $\mathbb{F}_q$ is the finite field with q elements.

Definition

Here, a hash function H maps tuples of elements of $\mathbb{F}_q$ to $\mathbb{F}_q^d$, for some fixed d.

Hash Functions

In what follows, $\mathbb{F}_q$ is the finite field with q elements.

Definition

Here, a hash function H maps tuples of elements of $\mathbb{F}_q$ to $\mathbb{F}_q^d$, for some fixed d.

Collision resistance: it must be infeasible in practice to find tuples x and y such that
$H(x) = H(y)$.

Oneway-ness: given $y \in (\mathbb{F}_q)^d$, it must be infeasible in practice to find x such that $H(x) = y$.

Hash Functions

In what follows, $\mathbb{F}_q$ is the finite field with q elements.

Definition

Here, a hash function H maps tuples of elements of $\mathbb{F}_q$ to $\mathbb{F}_q^d$, for some fixed d.

Collision resistance: it must be infeasible in practice to find tuples x and y such that $H(x) = H(y)$.

Oneway-ness: given $y \in (\mathbb{F}_q)^d$, it must be infeasible in practice to find x such that $H(x) = y$.

Examples

"Binary World"

- SHA-1 (broken)
- SHA-2
- SHA-3
- Whirlpool

Hash Functions

In what follows, $\mathbb{F}_q$ is the finite field with q elements.

Definition

Here, a hash function H maps tuples of elements of $\mathbb{F}_q$ to $\mathbb{F}_q^d$, for some fixed d.

Collision resistance: it must be infeasible in practice to find tuples x and y such that $H(x) = H(y)$.

Oneway-ness: given $y \in (\mathbb{F}_q)^d$, it must be infeasible in practice to find x such that $H(x) = y$.

Examples

"Binary World"

- SHA-1 (broken)
- SHA-2
- SHA-3
- Whirlpool

"Arithmetization-oriented"

- Rescue
- MiMC-hash
- gMiMC-hash
- Poseidon

A Natural Question

What are the differences between the “binary world” and the “arithmetization-oriented” world?

A Mismatch in Domain

For $\mathtt{SHA-X}$, we have

- $q = 2$
- $160 \leq d \leq 512$
- at least 10 years old
- Based on logical gates/CPU instructions

For arithmetization-oriented functions:

- $q \in \{2^n, p\}$, where $p \geq 2^n$, $n \geq 64$
- $2 \leq d \leq 4$
- at most 5 years old
- Based on finite field arithmetic

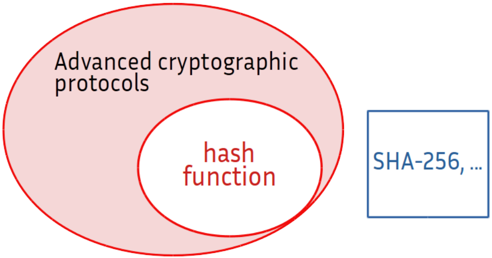
A Mismatch in Domain

For SHA−X, we have

- $q = 2$
- $160 \leq d \leq 512$
- at least 10 years old
- Based on logical gates/CPU instructions

For arithmetization-oriented functions:

- $q \in \{2^n, p\}$, where $p \geq 2^n$, $n \geq 64$
- $2 \leq d \leq 4$
- at most 5 years old
- Based on finite field arithmetic



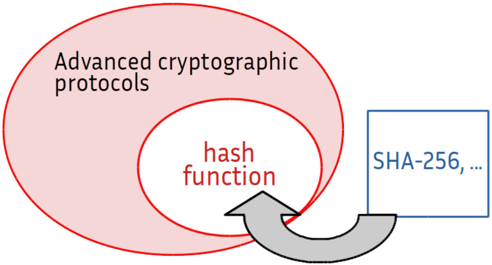
A Mismatch in Domain

For SHA−X, we have

- $q = 2$
- $160 \leq d \leq 512$
- at least 10 years old
- Based on logical gates/CPU instructions

For arithmetization-oriented functions:

- $q \in \{2^n, p\}$, where $p \geq 2^n$, $n \geq 64$
- $2 \leq d \leq 4$
- at most 5 years old
- Based on finite field arithmetic



A (Smaller) Mismatch in Properties

Binary Hash Functions

The sub-components must provide:

- **Security:** well-known attacks should not work
- **Operations:** $y \leftarrow R(x)$ must be fast/time constant
- **Efficiency:** easy implementation in software/hardware

A (Smaller) Mismatch in Properties

Binary Hash Functions

The sub-components must provide:

- **Security:** well-known attacks should not work
- **Operations:** $y \leftarrow R(x)$ must be fast/time constant
- **Efficiency:** easy implementation in software/hardware

Arithmetization-oriented Hash Functions

The sub-components must provide:

- **Security:** well-known attacks should not work
- **Operations:** verifying that $y = R(x)$ must be efficient
- **Efficiency:** easy integration without advanced protocols

A (Smaller) Mismatch in Properties

Binary Hash Functions

The sub-components must provide:

Security: well-known attacks should not work

Operations: $y \leftarrow R(x)$ must be fast/time constant

Efficiency: easy implementation in software/hardware

Arithmetization-oriented Hash Functions

The sub-components must provide:

Security: well-known attacks should not work

Operations: verifying that $y = R(x)$ must be efficient

Efficiency: easy integration without advanced protocols

A key difference: **indirect computation**

$$y \leftarrow R(x) \quad \text{vs.} \quad y == R(x)?$$

Take Away

Arithmetization-oriented functions differ **substantially** from “classical ones”!

Plan of this Section

1. What are Arithmetization-Oriented Hash Functions
   - Scope statement
   - How do we build and select symmetric primitives?
   - Examples of such Functions
2. How Do We Test Their Security?
3. Using CCZ-Equivalence to Outperform Everyone
4. Conclusion

To Build a Hash Function (Sponge Structure)

Modern hash functions usually have a
sponge structure

To Build a Hash Function (Sponge Structure)

Modern hash functions usually have a sponge structure

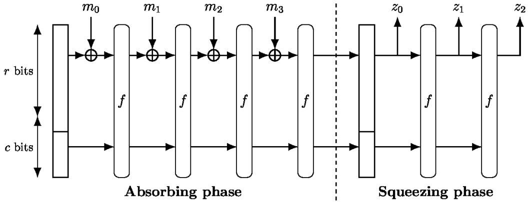


image source: `https://www.iacr.org/authors/tikz/`

Parameters:

- A **rate** $r > 0$ (≈ throughput)
- A **capacity** $c > 0$ (≈ security level)
- A public permutation f of $\mathbb{F}_q^r \times \mathbb{F}_q^c$.

Algorithm:

1. Turn the message into $(m_0, \ldots, m_{\ell-1})$, where $m_i \in \mathbb{F}_q^r$
2. Initialize $(x, y) \in \mathbb{F}_q^r \times \mathbb{F}_q^c$
3. For $i \in \{0, \ldots, \ell-1\}$:
   $$x \leftarrow x + m_i$$
   $$(x, y) \leftarrow f(x, y)$$
4. Return x

To Build a Hash Function (Round Function)

The main task is to build the permutation $f : X \mapsto Y$. **How** do we do this?

A **round function** R_i is iterated multiple times.

It is parameterized by the round number i.

To Build a Hash Function (Round Function)

The main task is to build the permutation $f : X \mapsto Y$. **How** do we do this?

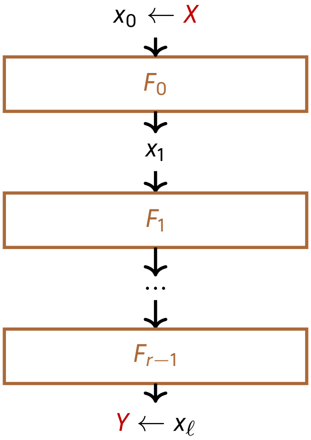
A **round function** R_i is iterated multiple times.

It is parameterized by the round number i.

How to build R_i?

The description of R_i is what really differentiates hash functions from one another.

(will be extensively discussed later)



To Build a Hash Function (Round Function)

The main task is to build the permutation $f : X \mapsto Y$. **How** do we do this?

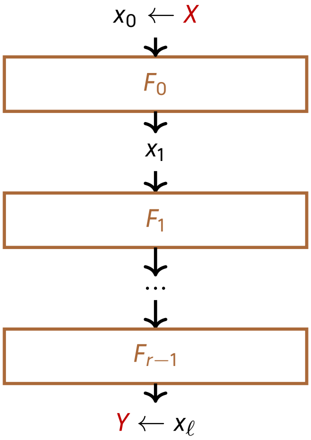
A **round function** R_i is iterated multiple times.
It is parameterized by the round number i.

How to build R_i?

The description of R_i is what really differentiates hash functions from one another.
(will be extensively discussed later)

How to choose the number r of rounds?

How many do we need to be safe from all known attacks, with some margin?
(a deep topic!)



Next step

OK, I have designed a round function R, chosen a number ℓ of rounds...

Next step

OK, I have designed a round function R, chosen a number ℓ of rounds...

Will people use my algorithm now?

Next step

OK, I have designed a round function R, chosen a number ℓ of rounds...

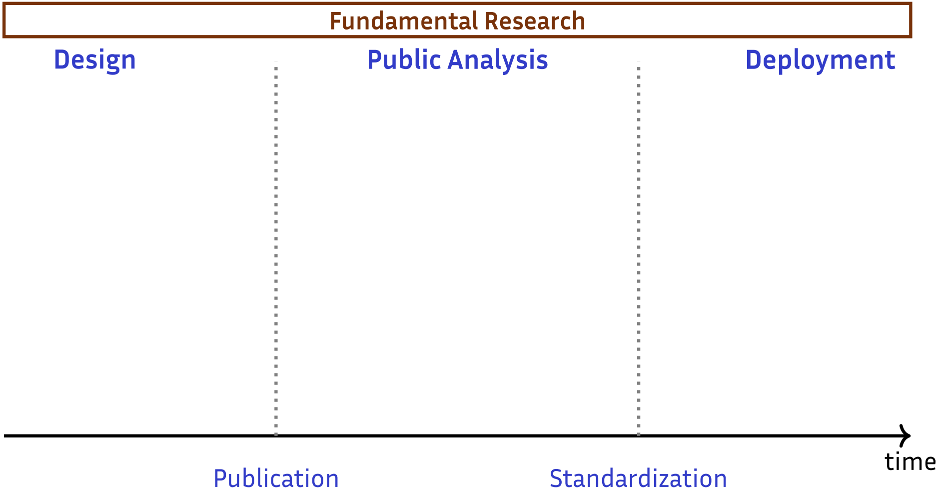
Will people use my algorithm now?

... No.

Cryptographic Pipeline

Fundamental Research

Cryptographic Pipeline



Cryptographic Pipeline

Fundamental Research

Design
Small teams

Public Analysis
Academic community

Deployment
Industry

time

Publication
Conf., competition

Standardization
NIST, ISO, IETF...

Cryptographic Pipeline

Fundamental Research

Design

Small teams

- Scope statement
- Algorithm specification
- Design choices justifications
- Security analysis

Public Analysis

Academic community

Deployment

Industry

time

Publication

Conf., competition

Standardization

NIST, ISO, IETF...

Cryptographic Pipeline

Fundamental Research

Design

Small teams

- Scope statement
- Algorithm specification
- Design choices justifications
- Security analysis

Public Analysis

Academic community

Deployment

Industry

time

Publication

Conf., competition

Standardization

NIST, ISO, IETF...

Cryptographic Pipeline

Fundamental Research

Design

Small teams

- Scope statement
- Algorithm specification
- Design choices justifications
- Security analysis

Public Analysis

Academic community

Try and break published algorithms

Deployment

Industry

time

Publication

Conf., competition

Standardization

NIST, ISO, IETF...

Cryptographic Pipeline

Fundamental Research

Design

Small teams

- Scope statement
- Algorithm specification
- Design choices justifications
- Security analysis

Public Analysis

Academic community

Try and break published algorithms

Deployment

Industry

time

Publication

Conf., competition

Standardization

NIST, ISO, IETF...

Cryptographic Pipeline

Fundamental Research

Design

Small teams

- Scope statement
- Algorithm specification
- Design choices justifications
- Security analysis

Public Analysis

Academic community

Try and break published algorithms

Unbroken algorithms are eventually trusted

Deployment

Industry

time

Publication

Conf., competition

Standardization

NIST, ISO, IETF...

Cryptographic Pipeline

Fundamental Research

Design

Small teams

- Scope statement
- Algorithm specification
- Design choices justifications
- Security analysis

Public Analysis

Academic community

Try and break published algorithms

Unbroken algorithms are eventually trusted

Deployment

Industry

Implements algorithms in actual products...

...unless a new attack is found

time

Publication

Conf., competition

Standardization

NIST, ISO, IETF...

Cryptographic Pipeline

Fundamental Research

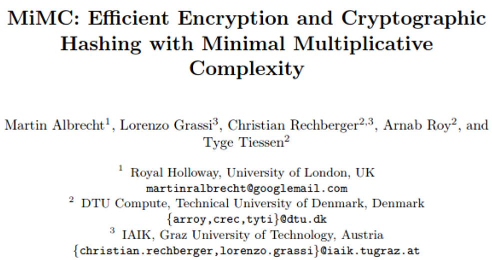
This process is slow, so we can have trust

Take Away

1. The adoption of new hash functions will depend on how much we trust them, and thus on their security arguments
2. These security arguments must be based on fundamental research

Plan of this Section

MiMC


MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity

Martin Albrecht[1], Lorenzo Grassi[3], Christian Rechberger[2,3], Arnab Roy[2], and Tyge Tiessen[2]

[1] Royal Holloway, University of London, UK
martinralbrecht@googlemail.com
[2] DTU Compute, Technical University of Denmark, Denmark
{arroy,crec,tyti}@dtu.dk
[3] IAIK, Graz University of Technology, Austria
{christian.rechberger,lorenzo.grassi}@iaik.tugraz.at

Published at ASIACRYPT'16;

https://eprint.iacr.org/2016/492.pdf

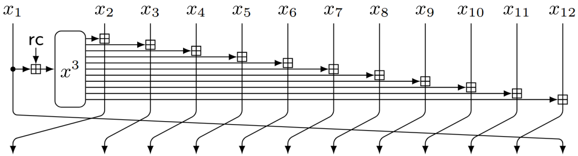
- Base field: $\mathbb{F}_q$, where e.g. $q = 2^{129}$
- Round function:

$$R_i \begin{cases} \mathbb{F}_q & \to \mathbb{F}_q \\ x & \mapsto (x + c_i)^3 \end{cases}$$

where the *round constants* c_i have been generated randomly.

- Number of rounds: $\ell \approx 90$

gMiMC

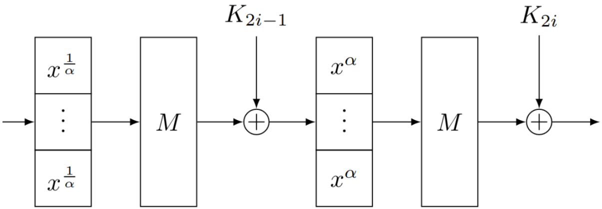


Published at ESORICS'19;
Albrecht, Perrin, Ramacher, Rechberger, Rotaru, Roy, Schofnegger

https://eprint.iacr.org/2019/397.pdf

- Base field: $\mathbb{F}_q$, where $q = 2^n$ or $q = p \geq 2^n$, $n \geq 64$
- Round function: see left
- Number of rounds: $\ell > 170$

Rescue



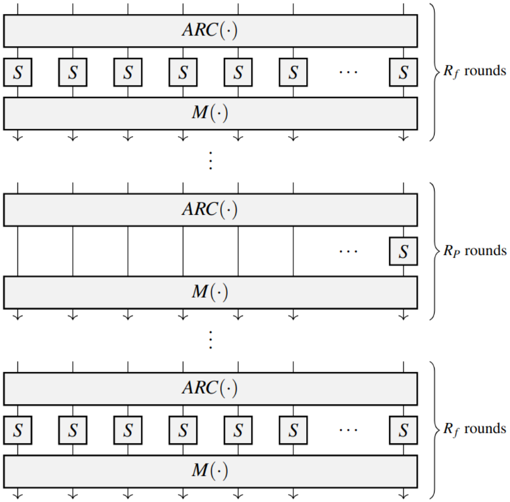
Published at ToSC'20(3);
Aly, Ashur, Ben-Sasson, Dhooghe, Szepieniec

https://tosc.iacr.org/index.php/ToSC/article/view/8695/8287

- Base field: $\mathbb{F}_q$, where $q = p \geq 2^n$, $n \geq 64$
- Round function: see left; $\alpha = 3$ and M is a linear permutation of $\mathbb{F}_q^t$.
- Number of rounds: $\ell \approx 10$

Verification: $P_i(x_i) == Q_i(x_{i+1})$, where P_i is a half round, and Q_i is the inverse of the other half!

Poseidon



Published at USENIX'21;
Grassi, Khovratovich, Rechberger, Roy, Schofnegger

`https://eprint.iacr.org/2019/458.pdf`

- Base field: $\mathbb{F}_q$, where $q = p \geq 2^n, n \geq 64$
- Round function: $S(x) = x^3$, ARC add a round constant, and M is a linear permutation of $\mathbb{F}_q^t$.
- Number of rounds: $\ell = R_f + R_P \approx 50$

Plan of this Section

Plan of this Section

Generic Attacks

Let H be a hash function with an output in $\mathbb{F}_q{}^d$.

Generic Attacks

Let H be a hash function with an output in ${\mathbb{F}_q}^d$.

No matter how good H is...

1. ... it can be inverted in time q^d (on average); (brute-force)
2. ... we can find x and y such that $H(x) = H(y)$ in time $\sqrt{q^d}$ (on average). (birthday search)

Generic attacks (such as these) serve as the benchmark
to assess security levels in symmetric cryptography.

Goal

What does it *mean* to attack a hash function?

Goal

What does it *mean* to attack a hash function?

Practical Attack

Actually exhibit x and y such that $H(x) = H(y)$.

Practically broken hash functions:

- MD4
- SHA-1

Goal

What does it *mean* to attack a hash function?

Practical Attack

Actually exhibit x and y such that $H(x) = H(y)$.

Practically broken hash functions:

- MD4
- SHA-1

Theoretical Result

Aim. Describe an algorithm capable of finding (x, y) faster than the corresponding generic attack.

Target. At first, we reduce the number of rounds in the inner primitive.

Goal

What does it *mean* to attack a hash function?

Practical Attack

Actually exhibit x and y such that $H(x) = H(y)$.

Practically broken hash functions:

- MD4
- SHA-1

Theoretical Result

Aim. Describe an algorithm capable of finding (x, y) faster than the corresponding generic attack.

Target. At first, we reduce the number of rounds in the inner primitive.

1. practical attacks are found after theoretical results

Goal

What does it *mean* to attack a hash function?

Practical Attack

Actually exhibit x and y such that $H(x) = H(y)$.

Practically broken hash functions:

- MD4
- SHA-1

Theoretical Result

Aim. Describe an algorithm capable of finding (x, y) faster than the corresponding generic attack.

Target. At first, we reduce the number of rounds in the inner primitive.

1. practical attacks are found after theoretical results
2. theoretical results on hash functions are found after theoretical results on its inner primitive (e.g. the permutation for sponge functions).

Milestone Towards the Goal

What does it *mean* to attack a **permutation**?

Milestone Towards the Goal

What does it *mean* to attack a **permutation**?

Does it even make sense?

The specification of a permutation is public: there is no **key** to protect!

- Ideally, an attacker wants to be able to control the **capacity** of the output using only the **rate** of the input.
- The security proof of the sponge relies on the permutation "behaving like a random permutation".

Milestone Towards the Goal

What does it *mean* to attack a **permutation**?

Does it even make sense?

The specification of a permutation is public: there is no **key** to protect!

- Ideally, an attacker wants to be able to control the **capacity** of the output using only the **rate** of the input.
- The security proof of the sponge relies on the permutation "behaving like a random permutation".

Examples of distinguishers

CICO. Can you find $(x, 0)$ such that $P(x, 0) = (y, 0)$ (faster than a brute-force search)?

Low Degree. The univariate (or algebraic) degree of P is lower than expected.

Differential. next slide

Others! Linear, integral...

Plan of this Section

Differential Attacks

Differential equation

$$P(x + a) - P(x) = b$$

Differential Attacks

Differential equation

$$P(x+a) - P(x) = b$$

- **Aim:** find (a, b) such that there are many solutions x.
- In practice, we find (a_i, a_{i+1}) at each round.

$$\begin{tikzcd}
x_0 = x \arrow[d, "F_0"'] & x_0 + a_0 = x + a \arrow[d, "F_0"'] \\
x_1 \arrow[d, "F_1"'] & x_1 + a_1 \arrow[d, "F_1"'] \\
\cdots \arrow[d, "F_{\ell-1}"'] & \cdots \arrow[d, "F_{\ell-1}"'] \\
P(x) & P(x+a) = P(x) + b
\end{tikzcd}$$

Differential Attacks

Differential equation

$$P(x+a) - P(x) \ = \ b$$

- **Aim:** find (a, b) such that there are many solutions x.
- In practice, we find (a_i, a_{i+1}) at each round.
- Successfully applied to the inner block cipher of SHA−1 (in $\{0,1\}^*$), thus leading to its break...
- ... A priori less applicable in $\mathbb{F}_q$ (or is it? $\rightarrow$ RESCUE)

$$\begin{array}{ccc}
x_0 = x & & x_0 + a_0 = x + a \\
\downarrow F_0 & & \downarrow F_0 \\
x_1 & & x_1 + a_1 \\
\downarrow F_1 & & \downarrow F_1 \\
\cdots & & \cdots \\
\downarrow F_{\ell-1} & & \downarrow F_{\ell-1} \\
P(x) & & P(x+a) = P(x) + b
\end{array}$$

Algebraic Attacks

$$\begin{tikzcd}
x_0 = x \arrow[d, "F_0"'] \\
x_1 \arrow[d, "F_1"'] \\
\cdots \arrow[d, "F_{\ell-1}"'] \\
P(x) = x_\ell
\end{tikzcd}$$

Main equation system

$$\begin{cases} x_1 & = F_0(x_0) \\ \dots & \\ x_\ell & = F_{\ell-1}(x_{\ell-1}) \end{cases}$$

Algebraic Attacks

$$\begin{array}{c} x_0 = x \\ \Big\downarrow F_0 \\ x_1 \\ \Big\downarrow F_1 \\ \cdots \\ \Big\downarrow F_{\ell-1} \\ P(x) = x_\ell \end{array}$$

Main equation system

$$\begin{cases} x_1 & = F_0(x_0) \\ \dots & \\ x_\ell & = F_{\ell-1}(x_{\ell-1}) \end{cases}$$

- If the system can be solved, then we can enforce constraints on x_0 and x_ℓ (e.g. CICO).

Algebraic Attacks

$$
\begin{tikzcd}
x_0 = x \arrow[d, "F_0"'] \\
x_1 \arrow[d, "F_1"'] \\
\cdots \arrow[d, "F_{\ell-1}"'] \\
P(x) = x_\ell
\end{tikzcd}
$$

Main equation system

$$
\begin{cases}
x_1 & = F_0(x_0) \\
\ldots & \\
x_\ell & = F_{\ell-1}(x_{\ell-1})
\end{cases}
$$

- If the system can be solved, then we can enforce constraints on x_0 and x_ℓ (e.g. CICO).
- First, compute a Gröbner basis of the system. Then, deduce a solution in the correct field.
- Complexity is not so easy to estimate:
  - We can give bounds based on the best Gröbner basis algorithms...
  - ... but they don't take the shape of the system into account.

Plan of this Section

Main Reference

*New Design Techniques for Efficient Arithmetization-Oriented Hash Functions: **Anemoi** Permutations and **Jive** Compression Mode*

- **Clémence Bouvier**, Sorbonne University, Inria
- **Pierre Briaud**, Sorbonne University, Inria
- **Pyrros Chaidos**, National and Kapodistrian University of Athens
- **Léo Perrin**, Inria
- **Robin Salen**, Toposware
- **Vesselin Velichkov**, Clearmatics, University of Edinburgh
- **Danny Willems**, LIX, Nomadic Labs

`https://eprint.iacr.org/2022/840`

Plan of this Section

Definition of CCZ-Equivalence (1/2)

Definition (Affine-Equivalence)

F and G are *affine equivalent* if $G(x) = (B \circ F \circ A)(x)$, where A, B are affine permutations.

Definition of CCZ-Equivalence (1/2)

Definition (Affine-Equivalence)

F and G are *affine equivalent* if $G(x) = (B \circ F \circ A)(x)$, where A, B are affine permutations.

Definition (EA-Equivalence; EA-mapping)

F and G are *E(xtended) A(ffine) equivalent* if $G(x) = (B \circ F \circ A)(x) + C(x)$, where A, B, C are affine and A, B are permutations;

Definition of CCZ-Equivalence (1/2)

Definition (Affine-Equivalence)

F and G are *affine equivalent* if $G(x) = (B \circ F \circ A)(x)$, where A, B are affine permutations.

Definition (EA-Equivalence; EA-mapping)

F and G are *E(xtended) A(ffine) equivalent* if $G(x) = (B \circ F \circ A)(x) + C(x)$, where A, B, C are affine and A, B are permutations; so that

$$\underbrace{\left\{(x, G(x)), \forall x \in \mathbb{F}_2^n\right\}}_{\Gamma_G} = \begin{bmatrix} A^{-1} & 0 \\ CA^{-1} & B \end{bmatrix} \left(\underbrace{\left\{(x, F(x)), \forall x \in \mathbb{F}_2^n\right\}}_{\Gamma_F} \right).$$

Definition of CCZ-Equivalence (2/2)

Definition (CCZ-Equivalence)

$F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ and $G : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ are *C(arlet)-C(harpin)-Z(inoviev) equivalent* if

$$\Gamma_G = \left\{(x, G(x)), \forall x \in \mathbb{F}_2^n\right\} \; = \; \mathcal{L}\left(\left\{(x, F(x)), \forall x \in \mathbb{F}_2^n\right\}\right) = \mathcal{L}(\Gamma_F)\,,$$

where $\mathcal{L} : \mathbb{F}_2^{n+m} \rightarrow \mathbb{F}_2^{n+m}$ is an affine permutation.

Definition of CCZ-Equivalence (2/2)

Definition (CCZ-Equivalence)

$F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ and $G : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ are *C(arlet)-C(harpin)-Z(inoviev) equivalent* if

$$\Gamma_G = \left\{(x, G(x)), \forall x \in \mathbb{F}_2^n\right\} = \mathcal{L}\left(\left\{(x, F(x)), \forall x \in \mathbb{F}_2^n\right\}\right) = \mathcal{L}(\Gamma_F),$$

where $\mathcal{L} : \mathbb{F}_2^{n+m} \rightarrow \mathbb{F}_2^{n+m}$ is an affine permutation.

Remark

In general, the CCZ-equivalence class of F is reduced to its extended-affine class...

Definition of CCZ-Equivalence (2/2)

Definition (CCZ-Equivalence)

$F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ and $G : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ are *C(arlet)-C(harpin)-Z(inoviev) equivalent* if

$$\Gamma_G = \left\{(x, G(x)), \forall x \in \mathbb{F}_2^n\right\} = \mathcal{L}\left(\left\{(x, F(x)), \forall x \in \mathbb{F}_2^n\right\}\right) = \mathcal{L}(\Gamma_F)\,,$$

where $\mathcal{L} : \mathbb{F}_2^{n+m} \rightarrow \mathbb{F}_2^{n+m}$ is an affine permutation.

Remark

In general, the CCZ-equivalence class of F is reduced to its extended-affine class...

But not always, and CCZ-equivalence does **not** preserve the degree!

Plan of this Section

Performance Metric

Verifying that $y == R(x)$ must be efficient...

Performance Metric

Verifying that $y == R(x)$ must be efficient...

"Efficiency" 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. $y = t_6 + x$

Performance Metric

Verifying that $y == R(x)$ must be efficient...

"Efficiency" 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. $y = t_6 + x$

Performance Metric

Verifying that $y == R(x)$ must be efficient...

"Efficiency" 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 4 constaints

Scope Statement

Arithmetization-oriented symmetric primitive

- Efficient and secure
- Enable low degree verification
- Have, overall, a high degree

CCZ-equivalence to the Rescue!

Suppose that F and G are CCZ-equivalent, and that

$$\{ (x, G(x)) \mid x \in \mathbb{F}_q \} = \mathcal{L}\left(\{ (x, F(x)) \mid x \in \mathbb{F}_q \}\right)$$

[1] Aly, A. et al. *Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols*. ToSC 2020(3), 1–45.

CCZ-equivalence to the Rescue!

Suppose that F and G are CCZ-equivalent, and that

$$\{ (x, G(x)) \mid x \in \mathbb{F}_q \} = \mathcal{L} \left(\{ (x, F(x)) \mid x \in \mathbb{F}_q \} \right)$$

We can test $y == F(x)$, or, equivalently, we can do the following:

1. $(u, v) = \mathcal{L}(x, y)$
2. test $v == G(u)$

[1] Aly, A. et al. *Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols*. ToSC 2020(3), 1–45.

CCZ-equivalence to the Rescue!

Suppose that F and G are CCZ-equivalent, and that

$$\{ (x, G(x)) \mid x \in \mathbb{F}_q \} = \mathcal{L}\left(\{ (x, F(x)) \mid x \in \mathbb{F}_q \}\right)$$

We can test $y == F(x)$, or, equivalently, we can do the following:

1. $(u, v) = \mathcal{L}(x, y)$
2. test $v == G(u)$

This trick was already used implicitly in Rescue[1], where $F(x) = x^{1/d}$ and $G(x) = x^d$.

[1] Aly, A. et al. *Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols*. ToSC 2020(3), 1–45.

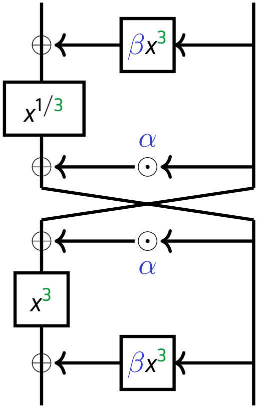
Scope Statement (more precise)

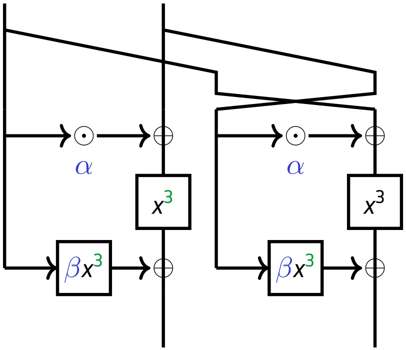
Arithmetization-oriented symmetric primitive

- Efficient and secure
- Based on high degree components that are CCZ-equivalent to low degree ones!

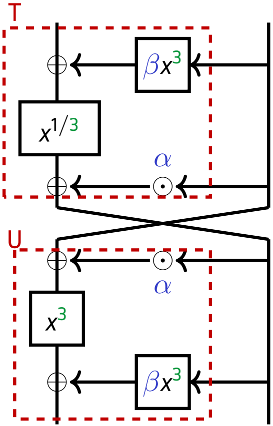
Plan of this Section

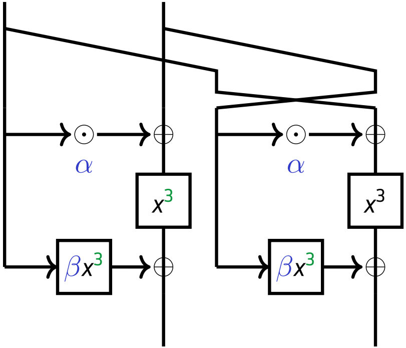
The Butterfly (reminder)



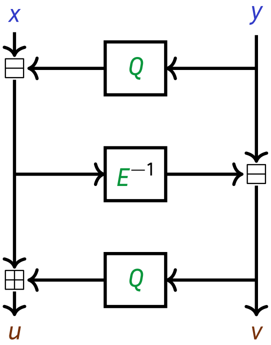


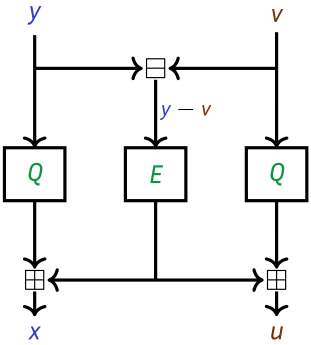
The Butterfly (reminder)





A Generalization of the case $\alpha = 1$: the Flystel

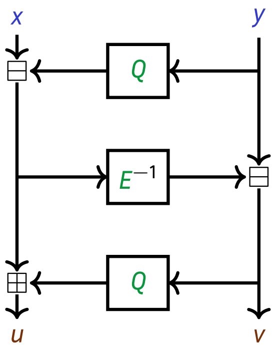


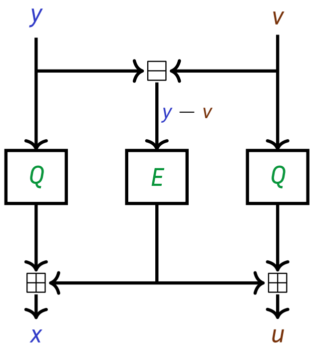


$q = 2^n$

$E(x) = x^3, Q(x) = \beta x^3$

A Generalization of the case $\alpha = 1$: the Flystel





$q = 2^n$

$E(x) = x^3$, $Q(x) = \beta x^3$

q prime

$E(x) = x^d$, $Q(x) = \beta x^2$

Properties of the Flystel

Theorem

A Flystel with $E(x) = x^d$ is differentially $(d-1)$-uniform.

Properties of the Flystel

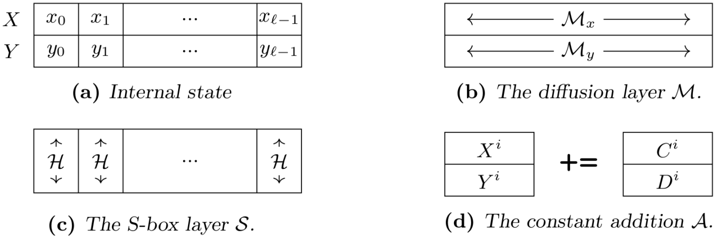
Theorem

A Flystel with $E(x) = x^d$ is differentially $(d-1)$-uniform.

Corollary

If $\gcd(q-1, 3) = 1$, then the open Flystel with $E(x) = x^3$ is an APN permutation of a field of even degree!

Anemoi



(a) *Internal state*

(b) *The diffusion layer $\mathcal{M}$.*

(c) *The S-box layer $\mathcal{S}$.*

(d) *The constant addition $\mathcal{A}$.*

Fig. 6: *The internal state of* `Anemoi` *and its basic operations.*

Performances (general)

	m	*Rescue'*	Poseidon	Griffin	Anemoi
R1CS	2	208	198	-	**76**
	3	216	214	**96**	-
	4	224	232	112	**96**
	6	216	264	-	**120**
	8	256	296	176	**160**
Plonk	2	312	380	-	**171**
	3	432	760	**214**	-
	4	560	1336	334	**216**
	6	756	3024	-	**330**
	8	1152	5448	969	**520**
AIR	2	156	300	-	**114**
	3	162	324	**144**	-
	4	168	348	168	**144**
	6	**162**	396	-	180
	8	**192**	480	264	240

(a) *when* $\alpha = 3$.

	m	*Rescue'*	Poseidon	Griffin	Anemoi
R1CS	2	240	216	-	**95**
	3	252	240	**96**	-
	4	264	264	**110**	120
	6	288	315	-	**150**
	8	384	363	**162**	200
Plonk	2	320	344	-	**190**
	3	420	624	**186**	-
	4	528	1032	287	**240**
	6	768	2265	-	**360**
	8	1280	4003	821	**560**
AIR	2	200	360	-	**190**
	3	210	405	**180**	-
	4	**220**	440	**220**	240
	6	**240**	540	-	300
	8	**320**	640	360	400

(b) *when* $\alpha = 5$.

Table 4: *Constraint comparison for several hash functions. We fix* $s = 128$.

Performances (general)

Table 2: *Constraints comparison of several hash functions for Plonk with an additional custom gate to compute x^5. We fix $s = 128$, and prime field sizes of 256.*

	m	Constraints
Poseidon	3	110
	2	88
Reinforced Concrete	3	378
	2	236
Griffin	3	125
AnemoiJive	2	**79**

(a) *With 3 wires.*

	m	Constraints
Poseidon	3	98
	2	82
Reinforced Concrete	3	267
	2	174
Griffin	3	111
AnemoiJive	2	**58**

(b) *With 4 wires.*

Plan of this Section

Conclusion

- Designing airthmetization-oriented hash functions is difficult because it is largely uncharted territory...

Conclusion

- Designing airthmetization-oriented hash functions is difficult because it is largely uncharted territory...
- Attack **and** design need more sophisticated mathematics than before!
- There is room for improvement!

Conclusion

- Designing airthmetization-oriented hash functions is difficult because it is largely uncharted territory...
- Attack **and** design need more sophisticated mathematics than before!
- There is room for improvement!

Conclusion

- Designing airthmetization-oriented hash functions is difficult because it is largely uncharted territory...
- Attack **and** design need more sophisticated mathematics than before!
- There is room for improvement!

Thank you!