Designing hash functions in GF(q) is (much) harder than it looks

Léo Perrin[1]

including joint works with
Tim Beyne, Clémence Bouvier, Anne Canteaut, Itai Dinur, Maria Eichlseder,
Gregor Leander, Gaëtan Leurent, María Naya-Plasencia,

Yu Sasaki, Yosuke Todo, and Friedrich Wiemer

Inria, Paris

22nd November 2021

informatics mathematics

Conclusion

Changing the underlying mathematical structure in cryptographic primitives is a significant change that requires substantial care.

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 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

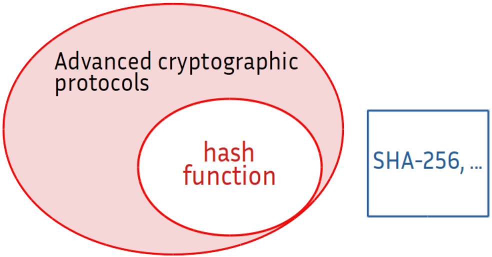
A Mismatch in Domain

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



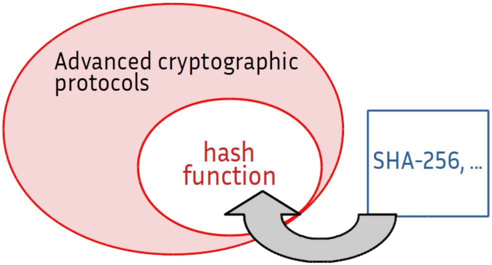
A Mismatch in Domain

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

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

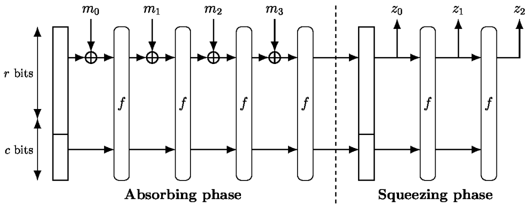


image source: `https://www.iacr.org/authors/tikz/`

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** F_i is iterated multiple times.
It is parameterized by the round number i.

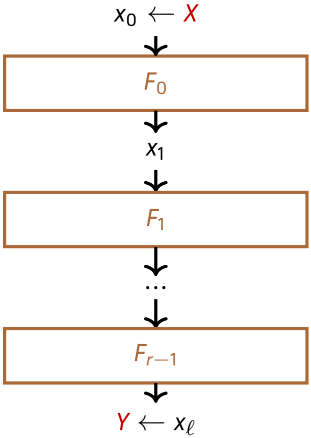
To Build a Hash Function (Round Function)

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

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

How to build F_i?

The description of F_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** F_i is iterated multiple times.

It is parameterized by the round number i.

How to build F_i?

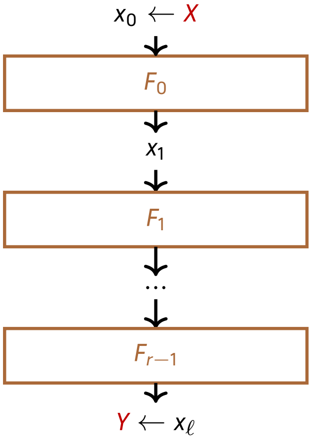
The description of F_i is what really differentiates hash functions from one another.

(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 F, chosen a number ℓ of rounds...

Next step

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

Will people use my algorithm now?

Next step

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

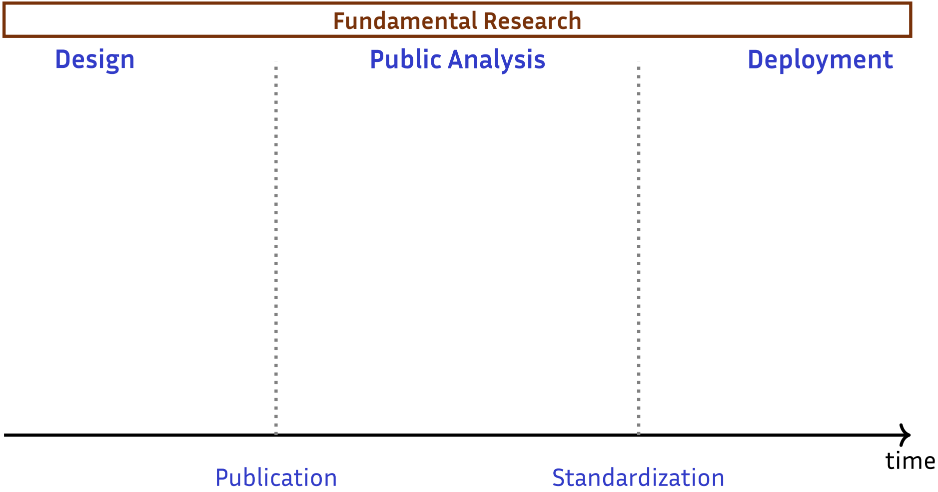
Will people use my algorithm now?

... No.

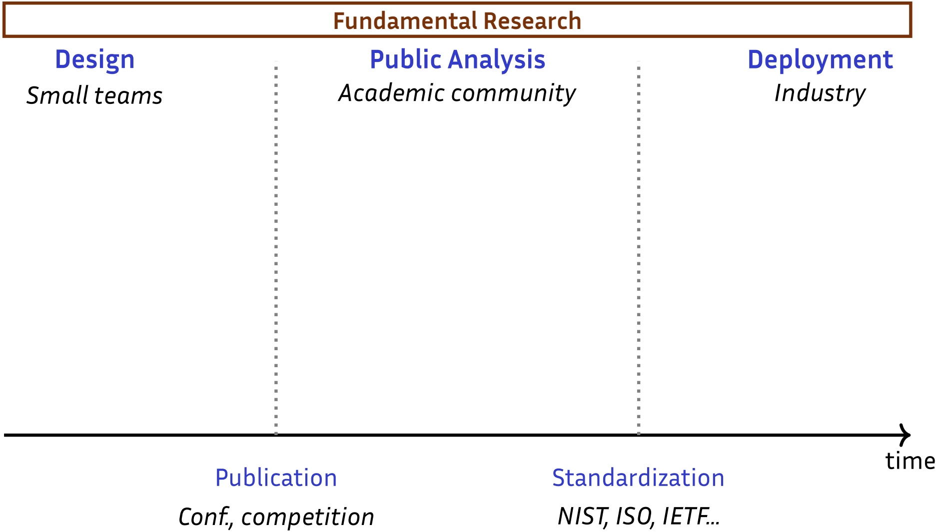
Cryptographic Pipeline

Fundamental Research

Cryptographic Pipeline



Cryptographic Pipeline



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

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. Some Cryptanalyses
4. Conclusion

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

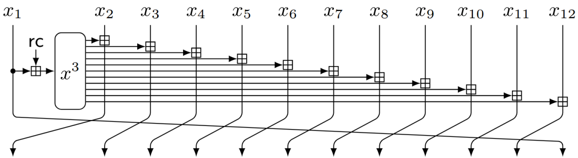
- Base field: $\mathbb{F}_q$, where e.g. $q = 2^{129}$
- Round function:

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

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

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

gMiMC

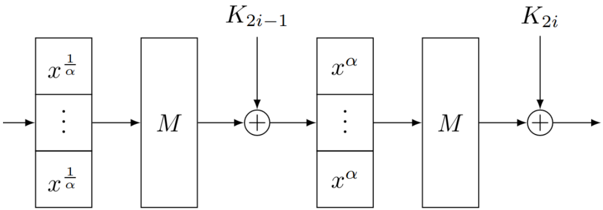


Published at ESORICS'19;
Albrecht, Perrin, Ramacher, Rechberger, Rotaru, Roy, Schofnegger

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



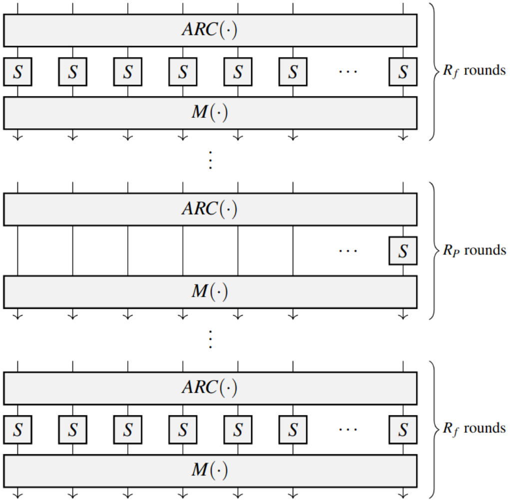
Published at ToSC'20(3);
Aly, Ashur, Ben-Sasson, Dhooghe, Szepieniec

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

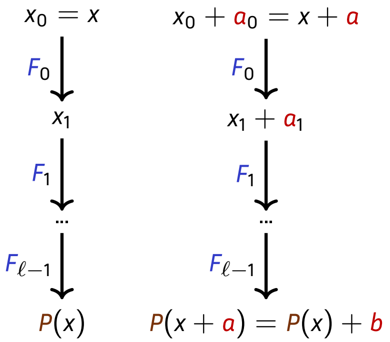
$$P(x + a) - P(x) = b$$

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.

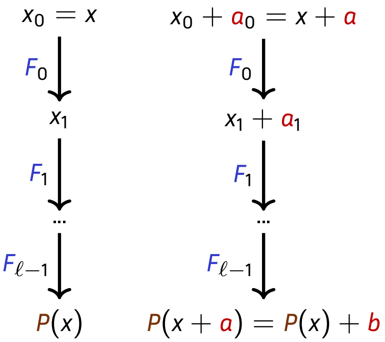


Differential Attacks

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)



Algebraic Attacks

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

Main equation system

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

Algebraic Attacks

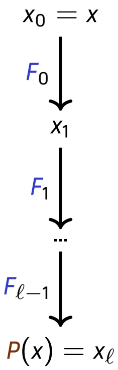
$$x_0 = x \xrightarrow{F_0} x_1 \xrightarrow{F_1} \cdots \xrightarrow{F_{\ell-1}} P(x) = x_\ell$$

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



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).
- 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

The limits of previous attempts

Algorithm		Attacks	
MiMC	ASIACRYPT'16	Higher-order differential	*in progress*
gMiMC	ESORICS'19	Integral attack	CRYPTO'20
Jarvis/RESCUE	ToSC'18	Algebraic attack	ASIACRYPT'19
		Differential attack	eprint 2020/820
Starkad/Poseidon	USENIX'21	Invariant subspace	CRYPTO'20, EUROCRYPT'21

Plan of this Section

Accurate Computations of the Algebraic Degree

Joint work with Clémence Bouvier and Anne Canteaut

Accurate Computations of the Algebraic Degree

Joint work with Clémence Bouvier and Anne Canteaut

Definition

Algebraic Degree $\mathbb{F}_{2^n}$ can be seen as $(\mathbb{F}_2)^n$, so
$G : \mathbb{F}_{2^n} \rightarrow \mathbb{F}_{2^n}$ is the same as
$G' : (\mathbb{F}_2)^n \rightarrow (\mathbb{F}_2)^n$, where

$$G'_i = \sum_{u \in (\mathbb{F}_2)^n} \alpha_u \prod_{i=0}^{n-1} x_i^{u_i}$$

The algebraic degree of G'_i is the maximum Hamming weight of u such that $\alpha_u \neq 0$.

- $\deg^a\left((x, y) \mapsto xy\right) = 2$
- $\deg^a(x \mapsto x^3) = 2$

Accurate Computations of the Algebraic Degree

Joint work with Clémence Bouvier and Anne Canteaut

Definition

Algebraic Degree $\mathbb{F}_{2^n}$ can be seen as $(\mathbb{F}_2)^n$, so
$G : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ is the same as
$G' : (\mathbb{F}_2)^n \to (\mathbb{F}_2)^n$, where

$$G'_i = \sum_{u \in (\mathbb{F}_2)^n} \alpha_u \prod_{i=0}^{n-1} x_i^{u_i}$$

The algebraic degree of G'_i is the maximum Hamming weight of u such that $\alpha_u \neq 0$.

- $\deg^a((x,y) \mapsto xy) = 2$
- $\deg^a(x \mapsto x^3) = 2$

The round function of MiMC is:

$$F_i : \begin{cases} \mathbb{F}_{2^n} & \to \mathbb{F}_{2^n} \\ x & \mapsto (x + c_i)^3 \end{cases}$$

The univariate degree is trivial (3^r), what about the algebraic degree?

Accurate Computations of the Algebraic Degree

Joint work with Clémence Bouvier and Anne Canteaut

Definition

Algebraic Degree $\mathbb{F}_{2^n}$ can be seen as $(\mathbb{F}_2)^n$, so
$G : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ is the same as
$G' : (\mathbb{F}_2)^n \to (\mathbb{F}_2)^n$, where

$$G'_i = \sum_{u \in (\mathbb{F}_2)^n} \alpha_u \prod_{i=0}^{n-1} x_i^{u_i}$$

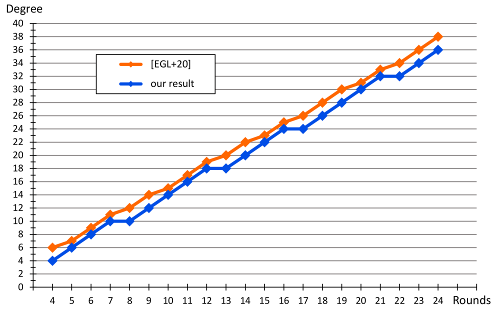
The algebraic degree of G'_i is the maximum Hamming weight of u such that $\alpha_u \neq 0$.

- $\deg^a((x, y) \mapsto xy) = 2$
- $\deg^a(x \mapsto x^3) = 2$

The round function of MiMC is:

$$F_i : \begin{cases} \mathbb{F}_{2^n} & \to \mathbb{F}_{2^n} \\ x & \mapsto (x + c_i)^3 \end{cases}$$

The univariate degree is trivial (3^r), what about the algebraic degree?



Take Away

Seemingly simple concepts have extremely complex behaviours in the arithmetization-friendly case!

Plan of this Section

Preliminary results on RESCUE

RESCUE is a block cipher published in 2019, which is the base of a hash function (RESCUE-prime).

Preliminary results on RESCUE

RESCUE is a block cipher published in 2019, which is the base of a hash function (RESCUE-prime).

Let $P : \mathbb{F}_q \rightarrow \mathbb{F}_q$. Its *differential uniformity* is:

$$\max_{a \neq 0, b \in \mathbb{F}_q} \# \left\{ x \in \mathbb{F}_q \mid P(x+a) - P(x) = b \right\} .$$

It must be low, and should decrease with the number of rounds/steps.

$\rightarrow$ Experimental verification for weakened variants of Rescue.

Preliminary results on RESCUE

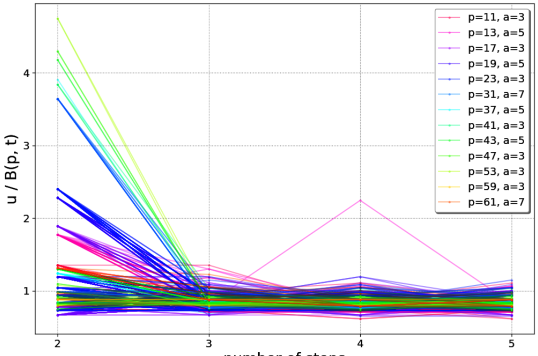
RESCUE is a block cipher published in 2019, which is the base of a hash function (RESCUE-prime).

Let $P : \mathbb{F}_q \rightarrow \mathbb{F}_q$. Its *differential uniformity* is:

$$\max_{a \neq 0, b \in \mathbb{F}_q} \# \{x \in \mathbb{F}_q \mid P(x+a) - P(x) = b\} \ .$$

It must be low, and should decrease with the number of rounds/steps.

$\rightarrow$ Experimental verification for weakened variants of Rescue.



Preliminary results on RESCUE

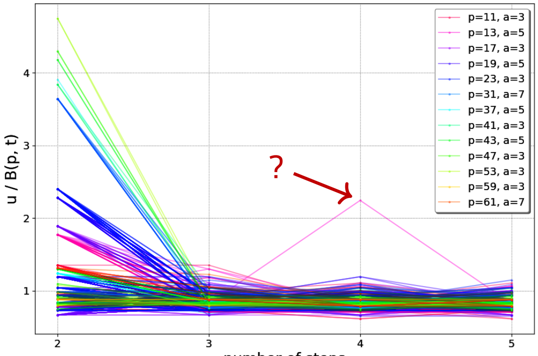
RESCUE is a block cipher published in 2019, which is the base of a hash function (RESCUE-prime).

Let $P : \mathbb{F}_q \rightarrow \mathbb{F}_q$. Its *differential uniformity* is:

$$\max_{a \neq 0, b \in \mathbb{F}_q} \# \{x \in \mathbb{F}_q \mid P(x+a) - P(x) = b\} \ .$$

It must be low, and should decrease with the number of rounds/steps.

$\rightarrow$ Experimental verification for weakened variants of Rescue.



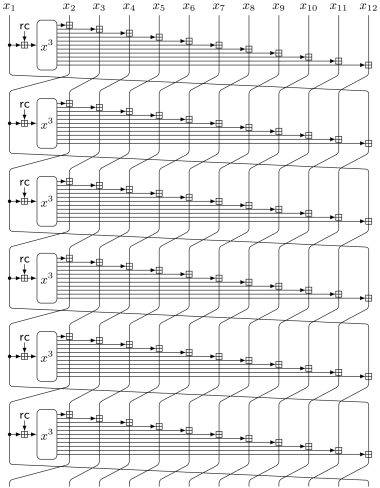
Take Away

This high probability differential *may* be a consequence of some multiplicative pattern traversing both $x \mapsto x^3$ and the linear layer.

No Frobenius automorphism implies that only simpler linear functions are available to designers, which in turn can imply strange differential behaviour!

Plan of this Section

Improving Integral Attacks

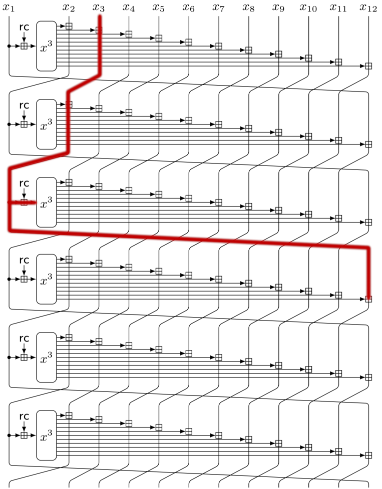


Principle (*saturation* approach)

1. Choose an input word, say, x_3.
2. Let it take all possible values while keeping other words constants.
3. Observe after each round whether each word is:

Constant	Sums to 0
Takes all possible values	Has no pattern anymore

Improving Integral Attacks



Principle (*saturation* approach)

1. Choose an input word, say, x_3.
2. Let it take all possible values while keeping other words constants.
3. Observe after each round whether each word is:

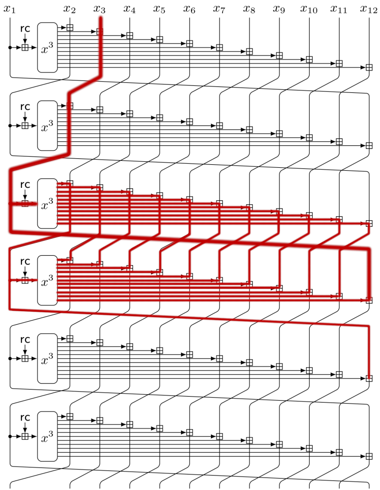
Constant

Takes all possible values

Sums to 0

Has no pattern anymore

Improving Integral Attacks



Principle (*saturation* approach)

1. Choose an input word, say, x_3.
2. Let it take all possible values while keeping other words constants.
3. Observe after each round whether each word is:

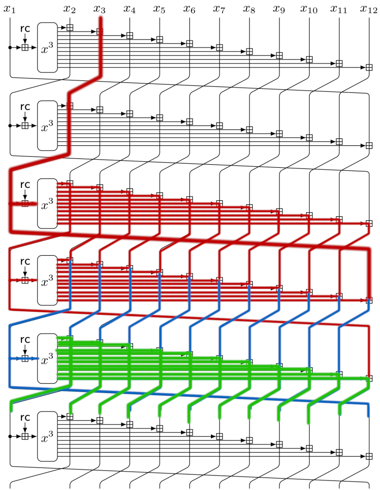
Constant

Sums to 0

Takes all possible values

Has no pattern anymore

Improving Integral Attacks

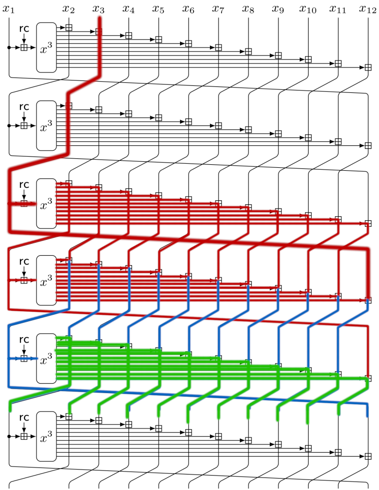


Principle (*saturation* approach)

1. Choose an input word, say, x_3.
2. Let it take all possible values while keeping other words constants.
3. Observe after each round whether each word is:

Constant	Sums to 0
Takes all possible values	Has no pattern anymore

Improving Integral Attacks



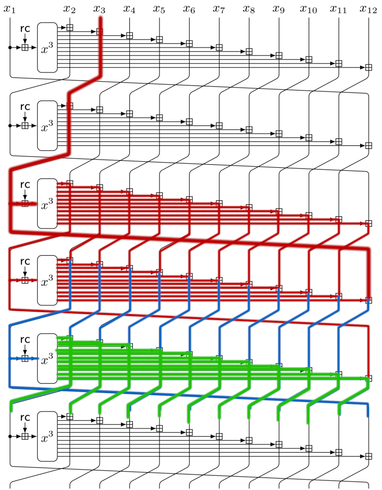
Principle (*saturation* approach)

1. Choose an input word, say, x_3.
2. Let it take all possible values while keeping other words constants.
3. Observe after each round whether each word is:

Constant	Sums to 0
Takes all possible values	Has no pattern anymore

Suppose that $q = 1 + \prod_i p_i$.
Then, there are many small multiplicative subgroups in $\mathbb{F}_q$,

Improving Integral Attacks



Principle (*saturation* approach)

1. Choose an input word, say, x_3.
2. Let it take all possible values while keeping other words constants.
3. Observe after each round whether each word is:

Constant	Sums to 0
Takes all possible values	Has no pattern anymore

Suppose that $q = 1 + \prod_i p_i$.
Then, there are many small multiplicative subgroups in $\mathbb{F}_q$, where all these properties are well defined!

Take Away

A basic saturation attack requires q queries to the permutation. If q is larger than the security parameter, they are infeasible.

The presence of small multiplicative subgroups significantly enhances saturation attacks by decreasing their data complexity!

Plan of this Section

Conclusion

1. Designing airthmetization-oriented hash functions is difficult because it is largely uncharted territory:
   - if $q = 2^n$, estimating the algebraic degree is hard (MiMC);
   - if $q = p$, multiplicative subgroups bother us (gMiMC);
   - if $q = p$, absence of Frobenius automorphisms $\rightarrow$ fewers options for linear operations (RESCUE).

Conclusion

1. Designing airthmetization-oriented hash functions is difficult because it is largely uncharted territory:
   - if $q = 2^n$, estimating the algebraic degree is hard (MiMC);
   - if $q = p$, multiplicative subgroups bother us (gMiMC);
   - if $q = p$, absence of Frobenius automorphisms $\rightarrow$ fewers options for linear operations (RESCUE).

2. There are theoretical issues with several of the new hash functions...

Conclusion

1. Designing airthmetization-oriented hash functions is difficult because it is largely uncharted territory:
   - if $q = 2^n$, estimating the algebraic degree is hard (MiMC);
   - if $q = p$, multiplicative subgroups bother us (gMiMC);
   - if $q = p$, absence of Frobenius automorphisms $\rightarrow$ fewers options for linear operations (RESCUE).
2. There are theoretical issues with several of the new hash functions...
3. There is room for improvement!

Conclusion

1. Designing airthmetization-oriented hash functions is difficult because it is largely uncharted territory:
   - if $q = 2^n$, estimating the algebraic degree is hard (MiMC);
   - if $q = p$, multiplicative subgroups bother us (gMiMC);
   - if $q = p$, absence of Frobenius automorphisms $\rightarrow$ fewers options for linear operations (RESCUE).

2. There are theoretical issues with several of the new hash functions...

3. There is room for improvement!

   Changing the underlying mathematical structure in cryptographic primitives is a significant change that requires substantial care.

Conclusion

1. Designing airthmetization-oriented hash functions is difficult because it is largely uncharted territory:
   - if $q = 2^n$, estimating the algebraic degree is hard (MiMC);
   - if $q = p$, multiplicative subgroups bother us (gMiMC);
   - if $q = p$, absence of Frobenius automorphisms $\rightarrow$ fewers options for linear operations (RESCUE).

2. There are theoretical issues with several of the new hash functions...

3. There is room for improvement!

   Changing the underlying mathematical structure in cryptographic primitives is a significant change that requires substantial care.

Thank you!