Flexible Modes for Arithmetization-Oriented Compression Functions

ALPSY 2025, January 25–29, Obergurgl

Stefano Trevisani

TU Wien Security & Privacy Group

Verifiable Computation, Blockchains, and ZK-SNARKs

Verifiable Computation and ZK-SNARKs

Verifiable Computation for Trusted Cloud/P2P:

Verifiable Computation and ZK-SNARKs

Verifiable Computation for Trusted Cloud/P2P:

- Server: computes some function $F(\mathrm{pub}, \mathrm{sec})$.

Verifiable Computation and ZK-SNARKs

Verifiable Computation for Trusted Cloud/P2P:

- Server: computes some function $F(\mathrm{pub}, \mathrm{sec})$.
- Client: verifies the correctness of the output.

Verifiable Computation and ZK-SNARKs

Verifiable Computation for Trusted Cloud/P2P:

- Server: computes some function $F(\mathrm{pub}, \mathrm{sec})$.
- Client: verifies the correctness of the output.
- ZK-SNARKs:
  - Server $\iff$ Prover, Client $\iff$ Verifier

Verifiable Computation and ZK-SNARKs

Verifiable Computation for Trusted Cloud/P2P:

- Server: computes some function $F(\mathrm{pub}, \mathrm{sec})$.
- Client: verifies the correctness of the output.
- ZK-SNARKs:
  - Server $\iff$ Prover, Client $\iff$ Verifier
- Virtual Machines, Blockchains, Recursive SNARKs...

Hash functions and ZK-SNARKs

Hash functions play a central role in SNARKs:

Hash functions and ZK-SNARKs

Hash functions play a central role in SNARKs:

- Blockchain rollups use **Merkle Trees** (MT)...

Hash functions and ZK-SNARKs

Hash functions play a central role in SNARKs:

- Blockchain rollups use **Merkle Trees** (MT)...
- ...And so do recursive SNARKs.

Hash functions and ZK-SNARKs

Hash functions play a central role in SNARKs:

- Blockchain rollups use **Merkle Trees** (MT)...
- ...And so do recursive SNARKs.
- ...And the FRI-based PCS used in STARK as well.

Hash functions and ZK-SNARKs

Hash functions play a central role in SNARKs:

- Blockchain rollups use **Merkle Trees** (MT)...
- ...And so do recursive SNARKs.
- ...And the FRI-based PCS used in STARK as well.
- Fiat-Shamir transform for non-interactive arguments.

Hash functions and ZK-SNARKs

Hash functions play a central role in SNARKs:

- Blockchain rollups use **Merkle Trees** (MT)...
- ...And so do recursive SNARKs.
- ...And the FRI-based PCS used in STARK as well.
- Fiat-Shamir transform for non-interactive arguments.

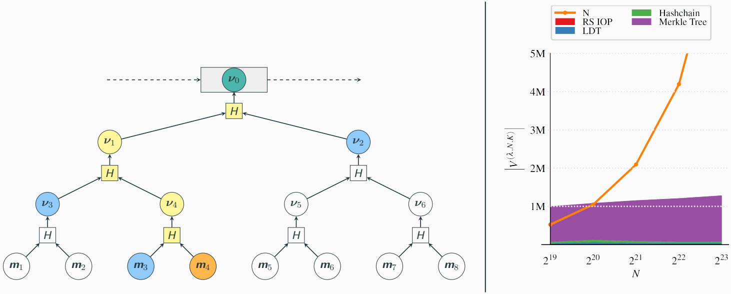


Figure 1: Left: binary Merkle Tree. Right: Fractal [12] verifier.

Arithmetization-Oriented Hash Functions

Cost of hashing in ZK-SNARK:

Cost of hashing in ZK-SNARK:

Arithmetization-Oriented Hash Functions

Cost of hashing in ZK-SNARK:

- Verification is fast (often constant time).

Arithmetization-Oriented Hash Functions

Cost of hashing in ZK-SNARK:

- Verification is fast (often constant time).
- Generation depends on **Multiplicative Complexity**:
  - Circuit over a **large** prime field $\mathbb{F}_p$ $(\log(p) \in \{64, 128, 256\})$.

Arithmetization-Oriented Hash Functions

Cost of hashing in ZK-SNARK:

- Verification is fast (often constant time).
- Generation depends on **Multiplicative Complexity**:
  - ◇ Circuit over a **large** prime field $\mathbb{F}_p$ $(\log(p) \in \{64, 128, 256\})$.
- Bit-oriented hash functions $\Rightarrow$ high MC.
  - ◇ Bitwise operations are expensive to emulate.

Arithmetization-Oriented Hash Functions

Cost of hashing in ZK-SNARK:

- Verification is fast (often constant time).
- Generation depends on **Multiplicative Complexity**:
  - ◇ Circuit over a **large** prime field $\mathbb{F}_p$ $(\log(p) \in \{64, 128, 256\})$.
- Bit-oriented hash functions $\Rightarrow$ high MC.
  - ◇ Bitwise operations are expensive to emulate.
- Arithmetization-oriented hash functions $\Rightarrow$ low MC.
  - ◇ defined directly over $\mathbb{F}_p$.

Cost of hashing in ZK-SNARK:

- Verification is fast (often constant time).
- Generation depends on **Multiplicative Complexity**:
  - ◇ Circuit over a **large** prime field $\mathbb{F}_p$ $(\log(p) \in \{64, 128, 256\})$.
- Bit-oriented hash functions $\Rightarrow$ high MC.
  - ◇ Bitwise operations are expensive to emulate.
- Arithmetization-oriented hash functions $\Rightarrow$ low MC.
  - ◇ defined directly over $\mathbb{F}_p$.
- Native (SW/HW) performance is still important!

Flexible AO Compression Modes

Joint work E. Andreeva, R. Bhattacharyya, A. Roy

Compositional Paradigms

Hash functions from **provably secure compositional** paradigms:

Compositional Paradigms

Hash functions from **provably secure compositional** paradigms:

- Permutation-based Sponge mode [6]:

Compositional Paradigms

Hash functions from **provably secure compositional** paradigms:

- Permutation-based Sponge mode [6]:
  - 😃 Provably secure over $\mathbb{F}_2$ and $\mathbb{F}_p$ [7, 22]).

Compositional Paradigms

Hash functions from **provably secure compositional** paradigms:

- Permutation-based Sponge mode [6]:
  - 😃 Provably secure over $\mathbb{F}_2$ and $\mathbb{F}_p$ [7, 22]).
  - 🙁 Cannot use the key input to compress data.

Compositional Paradigms

Hash functions from **provably secure compositional** paradigms:

- Permutation-based Sponge mode [6]:
  - 😃 Provably secure over $\mathbb{F}_2$ and $\mathbb{F}_p$ [7, 22]).
  - 🙁 Cannot use the key input to compress data.
- Blockcipher-based PGV modes [25]:

Compositional Paradigms

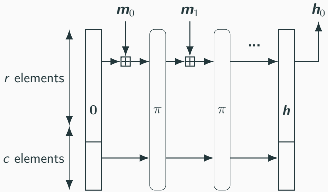
Hash functions from **provably secure compositional** paradigms:

- Permutation-based Sponge mode [6]:
  - 😃 Provably secure over $\mathbb{F}_2$ and $\mathbb{F}_p$ [7, 22]).
  - 🙁 Cannot use the key input to compress data.
- Blockcipher-based PGV modes [25]:
  - 🙂 Provably secure over $\mathbb{F}_2$ [9].

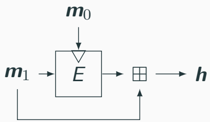
Compositional Paradigms

Hash functions from **provably secure compositional** paradigms:

- Permutation-based Sponge mode [6]:
  - 😃 Provably secure over $\mathbb{F}_2$ and $\mathbb{F}_p$ [7, 22]).
  - 🙁 Cannot use the key input to compress data.
- Blockcipher-based PGV modes [25]:
  - 🙂 Provably secure over $\mathbb{F}_2$ [9].
  - 😃 Exploits both key and plaintext inputs for compression.



Sponge



Davies-Meyer

Dealing with Input Length

Two kinds of hash modes:

Dealing with Input Length

Two kinds of hash modes:

- (Variable Input Length) Hash functions, inputs from $\mathbb{F}_p^*$:

Dealing with Input Length

Two kinds of hash modes:

- (Variable Input Length) Hash functions, inputs from $\mathbb{F}_p^*$:
  - 😃 Flexible input size.

Dealing with Input Length

Two kinds of hash modes:

- (Variable Input Length) Hash functions, inputs from $\mathbb{F}_p^*$:
  - 😃 Flexible input size.
  - 🙁 Require a padding scheme, suboptimal compression rate.

Dealing with Input Length

Two kinds of hash modes:

- (Variable Input Length) Hash functions, inputs from $\mathbb{F}_p^*$:
  - 😃 Flexible input size.
  - 🙁 Require a padding scheme, suboptimal compression rate.
  - 🙁 Wider attack surface (e.g. length extension attacks).

Dealing with Input Length

Two kinds of hash modes:

- (Variable Input Length) Hash functions, inputs from $\mathbb{F}_p^*$:
  - 😀 Flexible input size.
  - 🙁 Require a padding scheme, suboptimal compression rate.
  - 🙁 Wider attack surface (e.g. length extension attacks).
- *Compression functions*, inputs from $\mathbb{F}_p^m$:

Dealing with Input Length

Two kinds of hash modes:

- (Variable Input Length) Hash functions, inputs from $\mathbb{F}_p^*$:
  - 😀 Flexible input size.
  - 🙁 Require a padding scheme, suboptimal compression rate.
  - 🙁 Wider attack surface (e.g. length extension attacks).
- *Compression functions*, inputs from $\mathbb{F}_p^m$:
  - 😀 High compression rate.

Dealing with Input Length

Two kinds of hash modes:

- (Variable Input Length) Hash functions, inputs from $\mathbb{F}_p^*$:
  - Flexible input size.
  - Require a padding scheme, suboptimal compression rate.
  - Wider attack surface (e.g. length extension attacks).
- *Compression functions*, inputs from $\mathbb{F}_p^m$:
  - High compression rate.
  - 'Rigid' input size, usually rely on a small primitive.

Dealing with Input Length

Two kinds of hash modes:

- (Variable Input Length) Hash functions, inputs from $\mathbb{F}_p^*$:
  - 😃 Flexible input size.
  - 🙁 Require a padding scheme, suboptimal compression rate.
  - 🙁 Wider attack surface (e.g. length extension attacks).
- *Compression functions*, inputs from $\mathbb{F}_p^m$:
  - 😃 High compression rate.
  - 🙁 ‘Rigid’ input size, usually rely on a small primitive.

❓ Can we have the best of both?

Dealing with Input Length

Two kinds of hash modes:

- (Variable Input Length) Hash functions, inputs from $\mathbb{F}_p^*$:
  - 😃 Flexible input size.
  - 🙁 Require a padding scheme, suboptimal compression rate.
  - 🙁 Wider attack surface (e.g. length extension attacks).
- *Compression functions*, inputs from $\mathbb{F}_p^m$:
  - 😃 High compression rate.
  - 🙁 'Rigid' input size, usually rely on a small primitive.

❓ Can we have the best of both?

💡 Exploit flexibility of AO *design strategies*!

Dealing with Input Length

Two kinds of hash modes:

- (Variable Input Length) Hash functions, inputs from $\mathbb{F}_p^*$:
  - 😀 Flexible input size.
  - 🙁 Require a padding scheme, suboptimal compression rate.
  - 🙁 Wider attack surface (e.g. length extension attacks).
- *Compression functions*, inputs from $\mathbb{F}_p^m$:
  - 😀 High compression rate.
  - 🙁 'Rigid' input size, usually rely on a small primitive.

❓ Can we have the best of both?

💡 Exploit flexibility of AO *design strategies*!

The PGV-ELC **modes**

We introduced the PGV-ELC **family of modes**:

The PGV-ELC **modes**

We introduced the PGV-ELC **family of modes**:

- Published at IEEE CSF 2024 [2].

The PGV-ELC modes

We introduced the PGV-ELC **family of modes**:

- Published at IEEE CSF 2024 [2].
- Based on a block cipher $E\colon \mathbb{F}_p^{\kappa} \times \mathbb{F}_p^{n} \to \mathbb{F}_p^{n}$.

The PGV-ELC **modes**

We introduced the PGV-ELC **family of modes**:

- Published at IEEE CSF 2024 [2].
- Based on a block cipher $E\colon \mathbb{F}_p^{\kappa} \times \mathbb{F}_p^{n} \to \mathbb{F}_p^{n}$.
- $\mathcal{C}_{\boldsymbol{K},\boldsymbol{P},\boldsymbol{F},\boldsymbol{R},E}$ maps $\boldsymbol{x}_0 \parallel \boldsymbol{x}_1 \in \mathbb{F}_p^{\kappa'+n'}$ to $\boldsymbol{h} \in \mathbb{F}_p^{\ell}$, with $\ell \leq n'$.

The PGV-ELC modes

We introduced the PGV-ELC **family of modes**:

- Published at IEEE CSF 2024 [2].
- Based on a block cipher $E\colon \mathbb{F}_p^{\kappa} \times \mathbb{F}_p^{n} \to \mathbb{F}_p^{n}$.
- $\mathcal{C}_{\boldsymbol{K},\boldsymbol{P},\boldsymbol{F},\boldsymbol{R},E}$ maps $\boldsymbol{x}_0 \parallel \boldsymbol{x}_1 \in \mathbb{F}_p^{\kappa'+n'}$ to $\boldsymbol{h} \in \mathbb{F}_p^{\ell}$, with $\ell \leq n'$.
- Expansion matrices $\boldsymbol{K} \in \mathbb{F}_p^{\kappa \times \kappa'}$ and $\boldsymbol{P} \in \mathbb{F}_p^{n \times n'}$.

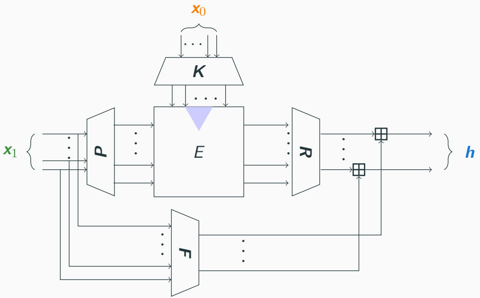
The PGV-ELC modes

We introduced the PGV-ELC **family of modes**:

- Published at IEEE CSF 2024 [2].
- Based on a block cipher $E\colon \mathbb{F}_p^{\kappa} \times \mathbb{F}_p^{n} \to \mathbb{F}_p^{n}$.
- $\mathcal{C}_{\boldsymbol{K},\boldsymbol{P},\boldsymbol{F},\boldsymbol{R},E}$ maps $\boldsymbol{x}_0 \parallel \boldsymbol{x}_1 \in \mathbb{F}_p^{\kappa'+n'}$ to $\boldsymbol{h} \in \mathbb{F}_p^{\ell}$, with $\ell \leq n'$.
- Expansion matrices $\boldsymbol{K} \in \mathbb{F}_p^{\kappa \times \kappa'}$ and $\boldsymbol{P} \in \mathbb{F}_p^{n \times n'}$.
- Compression matrices $\boldsymbol{F} \in \mathbb{F}_p^{\ell \times n'}$ and $\boldsymbol{R} \in \mathbb{F}_p^{\ell \times n}$.

We introduced the PGV-ELC **family of modes**:

- Published at IEEE CSF 2024 [2].
- Based on a block cipher $E \colon \mathbb{F}_p^{\kappa} \times \mathbb{F}_p^{n} \to \mathbb{F}_p^{n}$.
- $\mathcal{C}_{\boldsymbol{K},\boldsymbol{P},\boldsymbol{F},\boldsymbol{R},E}$ maps $\boldsymbol{x}_0 \parallel \boldsymbol{x}_1 \in \mathbb{F}_p^{\kappa'+n'}$ to $\boldsymbol{h} \in \mathbb{F}_p^{\ell}$, with $\ell \leq n'$.
- Expansion matrices $\boldsymbol{K} \in \mathbb{F}_p^{\kappa \times \kappa'}$ and $\boldsymbol{P} \in \mathbb{F}_p^{n \times n'}$.
- Compression matrices $\boldsymbol{F} \in \mathbb{F}_p^{\ell \times n'}$ and $\boldsymbol{R} \in \mathbb{F}_p^{\ell \times n}$.



The ELC-P **modes**

We also introduced the ELC-P **family of modes** [to appear]:

The ELC-P modes

We also introduced the ELC-P **family of modes** [to appear]:

- Based on a permutation $\pi\colon \mathbb{F}_p^m \to \mathbb{F}_p^m$.

The ELC-P **modes**

We also introduced the ELC-P **family of modes** [to appear]:

- Based on a permutation $\pi\colon \mathbb{F}_p^m \to \mathbb{F}_p^m$.
- $\mathcal{C}_{\boldsymbol{L},\boldsymbol{F},\boldsymbol{R},\pi}$ maps $\boldsymbol{x} \in \mathbb{F}_p^{m'}$ to $\boldsymbol{h} \in \mathbb{F}_p^{\ell}$, with $\ell \leq m'$.

The ELC-P modes

We also introduced the ELC-P **family of modes** [to appear]:

- Based on a permutation $\pi \colon \mathbb{F}_p^m \to \mathbb{F}_p^m$.
- $\mathcal{C}_{\boldsymbol{L},\boldsymbol{F},\boldsymbol{R},\pi}$ maps $\boldsymbol{x} \in \mathbb{F}_p^{m'}$ to $\boldsymbol{h} \in \mathbb{F}_p^{\ell}$, with $\ell \leq m'$.
- Expansion matrix $\boldsymbol{L} \in \mathbb{F}_p^{m \times m'}$.

The ELC-P modes

We also introduced the ELC-P **family of modes** [to appear]:

- Based on a permutation $\pi\colon \mathbb{F}_p^m \to \mathbb{F}_p^m$.
- $\mathcal{C}_{\boldsymbol{L},\boldsymbol{F},\boldsymbol{R},\pi}$ maps $\boldsymbol{x} \in \mathbb{F}_p^{m'}$ to $\boldsymbol{h} \in \mathbb{F}_p^{\ell}$, with $\ell \leq m'$.
- Expansion matrix $\boldsymbol{L} \in \mathbb{F}_p^{m \times m'}$.
- Compression matrices $\boldsymbol{F} \in \mathbb{F}_p^{\ell \times m'}$ and $\boldsymbol{R} \in \mathbb{F}_p^{\ell \times m}$.

The ELC-P modes

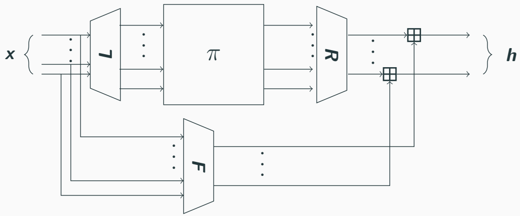
We also introduced the ELC-P **family of modes** [to appear]:

- Based on a permutation $\pi\colon \mathbb{F}_p^m \to \mathbb{F}_p^m$.
- $\mathcal{C}_{\boldsymbol{L},\boldsymbol{F},\boldsymbol{R},\pi}$ maps $\boldsymbol{x} \in \mathbb{F}_p^{m'}$ to $\boldsymbol{h} \in \mathbb{F}_p^{\ell}$, with $\ell \leq m'$.
- Expansion matrix $\boldsymbol{L} \in \mathbb{F}_p^{m \times m'}$.
- Compression matrices $\boldsymbol{F} \in \mathbb{F}_p^{\ell \times m'}$ and $\boldsymbol{R} \in \mathbb{F}_p^{\ell \times m}$.
- Includes existing modes like `Jive` [11] or Trunc [17, 19].

The ELC-P modes

We also introduced the ELC-P **family of modes** [to appear]:

- Based on a permutation $\pi\colon \mathbb{F}_p^m \to \mathbb{F}_p^m$.
- $\mathcal{C}_{\boldsymbol{L},\boldsymbol{F},\boldsymbol{R},\pi}$ maps $\boldsymbol{x} \in \mathbb{F}_p^{m'}$ to $\boldsymbol{h} \in \mathbb{F}_p^{\ell}$, with $\ell \leq m'$.
- Expansion matrix $\boldsymbol{L} \in \mathbb{F}_p^{m \times m'}$.
- Compression matrices $\boldsymbol{F} \in \mathbb{F}_p^{\ell \times m'}$ and $\boldsymbol{R} \in \mathbb{F}_p^{\ell \times m}$.
- Includes existing modes like `Jive` [11] or Trunc [17, 19].



Security Results

How to Prove Your Security

In order to **formally prove** that our modes are secure, we need:

How to Prove Your Security

In order to **formally prove** that our modes are secure, we need:

- A model for the underlying primitive(s) $\mathcal{P}$:
  - Ideal cipher/permutation $E \xleftarrow{\$} \mathrm{Block}\left(\mathbb{F}_p^{\kappa}, \mathbb{F}_p^{n}\right)$, $\pi \xleftarrow{\$} \mathrm{Perm}\left(\mathbb{F}_p^{m}\right)$.

How to Prove Your Security

In order to **formally prove** that our modes are secure, we need:

- A model for the underlying primitive(s) $\mathcal{P}$:
  - Ideal cipher/permutation $E \xleftarrow{\$} \mathrm{Block}\left(\mathbb{F}_p^{\kappa}, \mathbb{F}_p^{n}\right)$, $\pi \xleftarrow{\$} \mathrm{Perm}\left(\mathbb{F}_p^{m}\right)$.
- An adversary:
  - Query-bounded algorithm $\mathcal{A}$ with **oracle access** to $\mathcal{P}$.

How to Prove Your Security

In order to **formally prove** that our modes are secure, we need:

- A model for the underlying primitive(s) $\mathcal{P}$:
  - Ideal cipher/permutation $E \xleftarrow{\$} \mathrm{Block}\left(\mathbb{F}_p^{\kappa}, \mathbb{F}_p^{n}\right)$, $\pi \xleftarrow{\$} \mathrm{Perm}\left(\mathbb{F}_p^{m}\right)$.
- An adversary:
  - Query-bounded algorithm $\mathcal{A}$ with **oracle access** to $\mathcal{P}$.
- A security notion:
  - Collision/preimage resistance, indifferentiability, ...

How to Prove Your Security

In order to **formally prove** that our modes are secure, we need:

- A model for the underlying primitive(s) $\mathcal{P}$:
  - Ideal cipher/permutation $E \xleftarrow{\$} \mathsf{Block}\left(\mathbb{F}_p^\kappa, \mathbb{F}_p^n\right)$, $\pi \xleftarrow{\$} \mathsf{Perm}\left(\mathbb{F}_p^m\right)$.
- An adversary:
  - Query-bounded algorithm $\mathcal{A}$ with **oracle access** to $\mathcal{P}$.
- A security notion:
  - Collision/preimage resistance, indifferentiability, ...

Formalized by an advantage function:

- $\mathbf{Adv}_{\mathrm{mode}}^{\mathrm{NOTION}}(q) = \max_{\mathcal{A}}\{\mathbf{Adv}_{\mathrm{mode}}^{\mathrm{NOTION}}(\mathcal{A}, q)\}$.

How to Prove Your Security

In order to **formally prove** that our modes are secure, we need:

- A model for the underlying primitive(s) $\mathcal{P}$:
  - Ideal cipher/permutation $E \xleftarrow{\$} \mathsf{Block}(\mathbb{F}_p^\kappa, \mathbb{F}_p^n)$, $\pi \xleftarrow{\$} \mathsf{Perm}(\mathbb{F}_p^m)$.
- An adversary:
  - Query-bounded algorithm $\mathcal{A}$ with **oracle access** to $\mathcal{P}$.
- A security notion:
  - Collision/preimage resistance, indifferentiability, ...

Formalized by an advantage function:

- $\mathbf{Adv}_{\mathrm{mode}}^{\mathrm{NOTION}}(q) = \max_{\mathcal{A}}\{\mathbf{Adv}_{\mathrm{mode}}^{\mathrm{NOTION}}(\mathcal{A}, q)\}$.

Collision Resistance of PGV-ELC and ELC-P

Collision resistance of a compression mode $\mathcal{C}$:

$$\mathbf{Adv}_{\mathcal{C}}^{\mathrm{COL}}(\mathcal{A}, q) = \Pr\left[\left(\boldsymbol{x}, \boldsymbol{x}'\right) \xleftarrow{\$} \mathcal{A}^{\mathcal{P}}() : \boldsymbol{x} \neq \boldsymbol{x}' \wedge \mathcal{C}_{\mathcal{P}}(\boldsymbol{x}) = \mathcal{C}_{\mathcal{P}}\left(\boldsymbol{x}'\right)\right]$$

Collision Resistance of PGV-ELC and ELC-P

Collision resistance of a compression mode $\mathcal{C}$:

$$\mathbf{Adv}_{\mathcal{C}}^{\mathrm{COL}}(\mathcal{A}, q) = \Pr\left[\left(\boldsymbol{x}, \boldsymbol{x}'\right) \xleftarrow{\$} \mathcal{A}^{\mathcal{P}}() : \boldsymbol{x} \neq \boldsymbol{x}' \wedge \mathcal{C}_{\mathcal{P}}(\boldsymbol{x}) = \mathcal{C}_{\mathcal{P}}\left(\boldsymbol{x}'\right)\right]$$

For PGV-ELC: $\mathcal{C}_{E}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot E_{\boldsymbol{K}\boldsymbol{y}}(\boldsymbol{P}\boldsymbol{x}) + \boldsymbol{F}\boldsymbol{x}$:

Collision Resistance of PGV-ELC and ELC-P

Collision resistance of a compression mode $\mathcal{C}$:

$$\mathbf{Adv}_{\mathcal{C}}^{\mathrm{COL}}(\mathcal{A}, q) = \Pr\left[\left(\boldsymbol{x}, \boldsymbol{x}'\right) \xleftarrow{\$} \mathcal{A}^{\mathcal{P}}() : \boldsymbol{x} \neq \boldsymbol{x}' \wedge \mathcal{C}_{\mathcal{P}}(\boldsymbol{x}) = \mathcal{C}_{\mathcal{P}}\left(\boldsymbol{x}'\right)\right]$$

For PGV-ELC: $\mathcal{C}_E(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot E_{\boldsymbol{K}\boldsymbol{y}}(\boldsymbol{P}\boldsymbol{x}) + \boldsymbol{F}\boldsymbol{x}$:

1. Consider $\boldsymbol{R}, \boldsymbol{F}$ right-invertible, $\boldsymbol{K}, \boldsymbol{P}$ left-invertible.

Collision Resistance of PGV-ELC and ELC-P

Collision resistance of a compression mode $\mathcal{C}$:

$$\mathbf{Adv}_{\mathcal{C}}^{\mathrm{COL}}(\mathcal{A}, q) = \Pr\left[\left(\boldsymbol{x}, \boldsymbol{x}'\right) \xleftarrow{\$} \mathcal{A}^{\mathcal{P}}() : \boldsymbol{x} \neq \boldsymbol{x}' \wedge \mathcal{C}_{\mathcal{P}}(\boldsymbol{x}) = \mathcal{C}_{\mathcal{P}}\left(\boldsymbol{x}'\right)\right]$$

For PGV-ELC: $\mathcal{C}_E(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot E_{\boldsymbol{Ky}}(\boldsymbol{Px}) + \boldsymbol{Fx}$:

1. Consider $\boldsymbol{R}, \boldsymbol{F}$ right-invertible, $\boldsymbol{K}, \boldsymbol{P}$ left-invertible.
2. Matrices induce partitions over the row/column span.

Collision Resistance of PGV-ELC and ELC-P

Collision resistance of a compression mode $\mathcal{C}$:

$$\mathbf{Adv}_{\mathcal{C}}^{\mathrm{COL}}(\mathcal{A}, q) = \Pr\left[(\boldsymbol{x}, \boldsymbol{x}') \xleftarrow{\$} \mathcal{A}^{\mathcal{P}}() : \boldsymbol{x} \neq \boldsymbol{x}' \wedge \mathcal{C}_{\mathcal{P}}(\boldsymbol{x}) = \mathcal{C}_{\mathcal{P}}(\boldsymbol{x}')\right]$$

For PGV-ELC: $\mathcal{C}_{E}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot E_{\boldsymbol{K}\boldsymbol{y}}(\boldsymbol{P}\boldsymbol{x}) + \boldsymbol{F}\boldsymbol{x}$:

1. Consider $\boldsymbol{R}, \boldsymbol{F}$ right-invertible, $\boldsymbol{K}, \boldsymbol{P}$ left-invertible.
2. Matrices induce partitions over the row/column span.
3. Feed-forward addition guarantees one-wayness.

Collision Resistance of PGV-ELC **and** ELC-P

Collision resistance of a compression mode $\mathcal{C}$:

$$\mathbf{Adv}_{\mathcal{C}}^{\mathrm{COL}}(\mathcal{A}, q) = \Pr\left[\left(\boldsymbol{x}, \boldsymbol{x}'\right) \xleftarrow{\$} \mathcal{A}^{\mathcal{P}}() : \boldsymbol{x} \neq \boldsymbol{x}' \wedge \mathcal{C}_{\mathcal{P}}(\boldsymbol{x}) = \mathcal{C}_{\mathcal{P}}\left(\boldsymbol{x}'\right)\right]$$

For PGV-ELC: $\mathcal{C}_{E}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot E_{\boldsymbol{K}\boldsymbol{y}}(\boldsymbol{P}\boldsymbol{x}) + \boldsymbol{F}\boldsymbol{x}$:

1. Consider $\boldsymbol{R}, \boldsymbol{F}$ right-invertible, $\boldsymbol{K}, \boldsymbol{P}$ left-invertible.
2. Matrices induce partitions over the row/column span.
3. Feed-forward addition guarantees one-wayness.
4. $\mathcal{A}$ can adaptively exploit partition imbalances.

Collision Resistance of PGV-ELC and ELC-P

Collision resistance of a compression mode $\mathcal{C}$:

$$\mathbf{Adv}_{\mathcal{C}}^{\mathrm{COL}}(\mathcal{A}, q) = \Pr\left[\left(\boldsymbol{x}, \boldsymbol{x}'\right) \xleftarrow{\$} \mathcal{A}^{\mathcal{P}}() : \boldsymbol{x} \neq \boldsymbol{x}' \wedge \mathcal{C}_{\mathcal{P}}(\boldsymbol{x}) = \mathcal{C}_{\mathcal{P}}\left(\boldsymbol{x}'\right)\right]$$

For PGV-ELC: $\mathcal{C}_{E}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot E_{\boldsymbol{K}\boldsymbol{y}}(\boldsymbol{P}\boldsymbol{x}) + \boldsymbol{F}\boldsymbol{x}$:

1. Consider $\boldsymbol{R}, \boldsymbol{F}$ right-invertible, $\boldsymbol{K}, \boldsymbol{P}$ left-invertible.
2. Matrices induce partitions over the row/column span.
3. Feed-forward addition guarantees one-wayness.
4. $\mathcal{A}$ can adaptively exploit partition imbalances.
5. Still, we obtain $\mathbf{Adv}_{\mathcal{C}}^{\mathrm{COL}}(q) \leq \frac{q^2+q}{p^{\ell}-q}$ ($\approx$ birthday attack).

Collision Resistance of PGV-ELC and ELC-P

Collision resistance of a compression mode $\mathcal{C}$:

$$\mathbf{Adv}_{\mathcal{C}}^{\mathrm{COL}}(\mathcal{A}, q) = \Pr\left[\left(\boldsymbol{x}, \boldsymbol{x}'\right) \xleftarrow{\$} \mathcal{A}^{\mathcal{P}}() : \boldsymbol{x} \neq \boldsymbol{x}' \wedge \mathcal{C}_{\mathcal{P}}(\boldsymbol{x}) = \mathcal{C}_{\mathcal{P}}\left(\boldsymbol{x}'\right)\right]$$

For PGV-ELC: $\mathcal{C}_{E}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot E_{\boldsymbol{K}\boldsymbol{y}}(\boldsymbol{P}\boldsymbol{x}) + \boldsymbol{F}\boldsymbol{x}$:

1. Consider $\boldsymbol{R}, \boldsymbol{F}$ right-invertible, $\boldsymbol{K}, \boldsymbol{P}$ left-invertible.
2. Matrices induce partitions over the row/column span.
3. Feed-forward addition guarantees one-wayness.
4. $\mathcal{A}$ can adaptively exploit partition imbalances.
5. Still, we obtain $\mathbf{Adv}_{\mathcal{C}}^{\mathrm{COL}}(q) \leq \frac{q^2+q}{p^\ell - q}$ ($\approx$ birthday attack).

✅ Similar reasoning for ELC-P modes, preimage resistance.

Collision Resistance of PGV-ELC and ELC-P

Collision resistance of a compression mode $\mathcal{C}$:

$$\mathbf{Adv}_{\mathcal{C}}^{\mathrm{COL}}(\mathcal{A}, q) = \Pr\left[(\boldsymbol{x}, \boldsymbol{x}') \xleftarrow{\$} \mathcal{A}^{\mathcal{P}}() : \boldsymbol{x} \neq \boldsymbol{x}' \wedge \mathcal{C}_{\mathcal{P}}(\boldsymbol{x}) = \mathcal{C}_{\mathcal{P}}(\boldsymbol{x}')\right]$$

For PGV-ELC: $\mathcal{C}_E(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot E_{\boldsymbol{Ky}}(\boldsymbol{Px}) + \boldsymbol{Fx}$:

1. Consider $\boldsymbol{R}, \boldsymbol{F}$ right-invertible, $\boldsymbol{K}, \boldsymbol{P}$ left-invertible.
2. Matrices induce partitions over the row/column span.
3. Feed-forward addition guarantees one-wayness.
4. $\mathcal{A}$ can adaptively exploit partition imbalances.
5. Still, we obtain $\mathbf{Adv}_{\mathcal{C}}^{\mathrm{COL}}(q) \leq \frac{q^2+q}{p^\ell - q}$ ($\approx$ birthday attack).

✅ Similar reasoning for ELC-P modes, preimage resistance.

How to choose your matrix?

Consider ELC-P: $\mathcal{C}_{\pi}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot \pi(\boldsymbol{L}\boldsymbol{x}) + \boldsymbol{F}\boldsymbol{x}$:

How to choose your matrix?

Consider ELC-P: $\mathcal{C}_{\pi}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot \pi(\boldsymbol{L}\boldsymbol{x}) + \boldsymbol{F}\boldsymbol{x}$:

- How to choose the matrix $\boldsymbol{R}$?

How to choose your matrix?

Consider ELC-P: $\mathcal{C}_\pi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot \pi(\boldsymbol{L}\boldsymbol{x}) + \boldsymbol{F}\boldsymbol{x}$:

- How to choose the matrix $\boldsymbol{R}$?
  - Any pseudo-invertible matrix will do.

How to choose your matrix?

Consider ELC-P: $\mathcal{C}_\pi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot \pi(\boldsymbol{Lx}) + \boldsymbol{Fx}$:

- How to choose the matrix $\boldsymbol{R}$?
  - Any pseudo-invertible matrix will do.
  - What if we weaken our model?

How to choose your matrix?

Consider ELC-P: $\mathcal{C}_\pi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot \pi(\boldsymbol{Lx}) + \boldsymbol{Fx}$:

- How to choose the matrix $\boldsymbol{R}$?
  - Any pseudo-invertible matrix will do.
  - What if we weaken our model?
- $\mathcal{A}$ has access to an oracle $\mathcal{O}_t$, with $\ell < t \leq m$:

How to choose your matrix?

Consider ELC-P: $\mathcal{C}_{\pi}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot \pi(\boldsymbol{Lx}) + \boldsymbol{Fx}$:

- How to choose the matrix $\boldsymbol{R}$?
  - ◇ Any pseudo-invertible matrix will do.
  - ◇ What if we weaken our model?
- $\mathcal{A}$ has access to an oracle $\mathcal{O}_t$, with $\ell < t \leq m$:
  - ◇ $\mathrm{Tr}\big(\pi\big(\mathcal{O}_t\big(\boldsymbol{v} \in \mathbb{F}_p^t\big)\big)\big) = \boldsymbol{v}$

How to choose your matrix?

Consider ELC-P: $\mathcal{C}_\pi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot \pi(\boldsymbol{Lx}) + \boldsymbol{Fx}$:

- How to choose the matrix $\boldsymbol{R}$?
  - Any pseudo-invertible matrix will do.
  - What if we weaken our model?
- $\mathcal{A}$ has access to an oracle $\mathcal{O}_t$, with $\ell < t \leq m$:
  - $\mathrm{Tr}\big(\pi\big(\mathcal{O}_t\big(\boldsymbol{v} \in \mathbb{F}_p^t\big)\big)\big) = \boldsymbol{v}$
- If $\boldsymbol{R}$ is pseudo-identity (i.e. truncation), easy to get collisions!

How to choose your matrix?

Consider ELC-P: $\mathcal{C}_\pi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot \pi(\boldsymbol{Lx}) + \boldsymbol{Fx}$:

- How to choose the matrix $\boldsymbol{R}$?
  - Any pseudo-invertible matrix will do.
  - What if we weaken our model?
- $\mathcal{A}$ has access to an oracle $\mathcal{O}_t$, with $\ell < t \leq m$:
  - $\mathrm{Tr}\big(\pi\big(\mathcal{O}_t\big(\boldsymbol{v} \in \mathbb{F}_p^t\big)\big)\big) = \boldsymbol{v}$
- If $\boldsymbol{R}$ is pseudo-identity (i.e. truncation), easy to get collisions!
  - Choose $\boldsymbol{R}$ MDS.

How to choose your matrix?

Consider ELC-P: $\mathcal{C}_\pi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot \pi(\boldsymbol{Lx}) + \boldsymbol{Fx}$:

- How to choose the matrix $\boldsymbol{R}$?
  - Any pseudo-invertible matrix will do.
  - What if we weaken our model?
- $\mathcal{A}$ has access to an oracle $\mathcal{O}_t$, with $\ell < t \leq m$:
  - $\mathrm{Tr}\big(\pi\big(\mathcal{O}_t\big(\boldsymbol{v} \in \mathbb{F}_p^t\big)\big)\big) = \boldsymbol{v}$
- If $\boldsymbol{R}$ is pseudo-identity (i.e. truncation), easy to get collisions!
  - Choose $\boldsymbol{R}$ MDS.
- Related: AES last round missing MixColumns [13].

How to choose your matrix?

Consider ELC-P: $\mathcal{C}_\pi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot \pi(\boldsymbol{Lx}) + \boldsymbol{Fx}$:

- How to choose the matrix $\boldsymbol{R}$?
  - Any pseudo-invertible matrix will do.
  - What if we weaken our model?
- $\mathcal{A}$ has access to an oracle $\mathcal{O}_t$, with $\ell < t \leq m$:
  - $\mathrm{Tr}\big(\pi\big(\mathcal{O}_t\big(\boldsymbol{v} \in \mathbb{F}_p^t\big)\big)\big) = \boldsymbol{v}$
- If $\boldsymbol{R}$ is pseudo-identity (i.e. truncation), easy to get collisions!
  - Choose $\boldsymbol{R}$ MDS.
- Related: AES last round missing MixColumns [13].

Random Oracle Indifferentiability

Sometimes collision/preimage resistance is not enough:

Random Oracle Indifferentiability

Sometimes collision/preimage resistance is not enough:

- *Indifferentiability* $\approx$ compositional indistinguishability.

Random Oracle Indifferentiability

Sometimes collision/preimage resistance is not enough:

- *Indifferentiability* $\approx$ compositional indistinguishability.
- (FIL) Random Oracle $H \xleftarrow{\$} \mathsf{Func}\big(\mathbb{F}_p^m, \mathbb{F}_p^\ell\big)$.

Random Oracle Indifferentiability

Sometimes collision/preimage resistance is not enough:

- *Indifferentiability* $\approx$ compositional indistinguishability.
- (FIL) Random Oracle $H \xleftarrow{\$} \mathsf{Func}\left(\mathbb{F}_p^m, \mathbb{F}_p^\ell\right)$.
- Simulator $\mathcal{S}$ must mimic the primitive $\mathcal{P}$.
  - Can query H, should be (query) efficient.

Random Oracle Indifferentiability

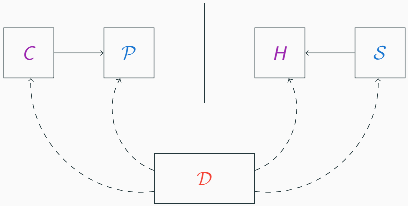
Sometimes collision/preimage resistance is not enough:

- *Indifferentiability* $\approx$ compositional indistinguishability.
- (FIL) Random Oracle $H \xleftarrow{\$} \mathrm{Func}(\mathbb{F}_p^m, \mathbb{F}_p^\ell)$.
- Simulator $\mathcal{S}$ must mimic the primitive $\mathcal{P}$.
  - Can query H, should be (query) efficient.
- Differentiator $\mathcal{D}$ must tell $(\mathcal{C}, \mathcal{P})$ apart from $(H, \mathcal{S})$.

Random Oracle Indifferentiability

Sometimes collision/preimage resistance is not enough:

- *Indifferentiability* $\approx$ compositional indistinguishability.
- (FIL) Random Oracle $H \xleftarrow{\$} \mathsf{Func}\left(\mathbb{F}_p^m, \mathbb{F}_p^\ell\right)$.
- Simulator $\mathcal{S}$ must mimic the primitive $\mathcal{P}$.
  - Can query H, should be (query) efficient.
- Differentiator $\mathcal{D}$ must tell $(\mathcal{C}, \mathcal{P})$ apart from $(H, \mathcal{S})$.



Indifferentiability of PGV-ELC and ELC-P

For ELC-P: $\mathcal{C}_{\pi}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot \pi(\boldsymbol{L}\boldsymbol{x}) + \boldsymbol{F}\boldsymbol{x}$:

Indifferentiability of PGV-ELC and ELC-P

For ELC-P: $\mathcal{C}_\pi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot \pi(\boldsymbol{L}\boldsymbol{x}) + \boldsymbol{F}\boldsymbol{x}$:

1. Again, $\boldsymbol{R}, \boldsymbol{F}$ right-invertible, $\boldsymbol{L}$ left-invertible.

Indifferentiability of PGV-ELC and ELC-P

For ELC-P: $\mathcal{C}_\pi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot \pi(\boldsymbol{L}\boldsymbol{x}) + \boldsymbol{F}\boldsymbol{x}$:

1. Again, $\boldsymbol{R}, \boldsymbol{F}$ right-invertible, $\boldsymbol{L}$ left-invertible.
2. We can devise $\mathcal{S}$ such that:

Indifferentiability of PGV-ELC and ELC-P

For ELC-P: $\mathcal{C}_\pi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot \pi(\boldsymbol{L}\boldsymbol{x}) + \boldsymbol{F}\boldsymbol{x}$:

1. Again, $\boldsymbol{R}, \boldsymbol{F}$ right-invertible, $\boldsymbol{L}$ left-invertible.
2. We can devise $\mathcal{S}$ such that:
   - Keeps track of queries coming from $\mathcal{D}$.

Indifferentiability of PGV-ELC and ELC-P

For ELC-P: $\mathcal{C}_\pi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot \pi(\boldsymbol{L}\boldsymbol{x}) + \boldsymbol{F}\boldsymbol{x}$:

1. Again, $\boldsymbol{R}, \boldsymbol{F}$ right-invertible, $\boldsymbol{L}$ left-invertible.
2. We can devise $\mathcal{S}$ such that:
   - Keeps track of queries coming from $\mathcal{D}$.
   - Makes (at most) one H-query per call.

Indifferentiability of PGV-ELC and ELC-P

For ELC-P: $\mathcal{C}_\pi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot \pi(\boldsymbol{Lx}) + \boldsymbol{Fx}$:

1. Again, $\boldsymbol{R}, \boldsymbol{F}$ right-invertible, $\boldsymbol{L}$ left-invertible.
2. We can devise $\mathcal{S}$ such that:
   - Keeps track of queries coming from $\mathcal{D}$.
   - Makes (at most) one H-query per call.
3. $\mathcal{D}$ can differentiate $\mathcal{S}$ from π via *backward queries*:

Indifferentiability of PGV-ELC and ELC-P

For ELC-P: $\mathcal{C}_\pi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot \pi(\boldsymbol{L}\boldsymbol{x}) + \boldsymbol{F}\boldsymbol{x}$:

1. Again, $\boldsymbol{R}, \boldsymbol{F}$ right-invertible, $\boldsymbol{L}$ left-invertible.
2. We can devise $\mathcal{S}$ such that:
   - Keeps track of queries coming from $\mathcal{D}$.
   - Makes (at most) one H-query per call.
3. $\mathcal{D}$ can differentiate $\mathcal{S}$ from π via *backward queries*:
   - $p^m - p^{m'}$ of them are *preimage-free* ($\boldsymbol{L}(\boldsymbol{L}^+\boldsymbol{y}) \neq \boldsymbol{y}$).

Indifferentiability of PGV-ELC and ELC-P

For ELC-P: $\mathcal{C}_\pi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot \pi(\boldsymbol{Lx}) + \boldsymbol{Fx}$:

1. Again, $\boldsymbol{R}, \boldsymbol{F}$ right-invertible, $\boldsymbol{L}$ left-invertible.
2. We can devise $\mathcal{S}$ such that:
   - Keeps track of queries coming from $\mathcal{D}$.
   - Makes (at most) one H-query per call.
3. $\mathcal{D}$ can differentiate $\mathcal{S}$ from π via *backward queries*:
   - $p^m - p^{m'}$ of them are *preimage-free* ($\boldsymbol{L}(\boldsymbol{L}^+\boldsymbol{y}) \neq \boldsymbol{y}$).
4. We can bound $\mathbf{Adv}_{\mathcal{C}}^{\mathrm{DIF}}(q) \leq \frac{q}{p^{m-m'}-q}$
   - q is the sum of primitive and construction queries.

For ELC-P: $\mathcal{C}_\pi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{R} \cdot \pi(\boldsymbol{Lx}) + \boldsymbol{Fx}$:

1. Again, $\boldsymbol{R}, \boldsymbol{F}$ right-invertible, $\boldsymbol{L}$ left-invertible.
2. We can devise $\mathcal{S}$ such that:
   - Keeps track of queries coming from $\mathcal{D}$.
   - Makes (at most) one H-query per call.
3. $\mathcal{D}$ can differentiate $\mathcal{S}$ from π via *backward queries*:
   - $p^m - p^{m'}$ of them are *preimage-free* $(\boldsymbol{L}(\boldsymbol{L}^+\boldsymbol{y}) \neq \boldsymbol{y})$.
4. We can bound $\mathbf{Adv}_{\mathcal{C}}^{\mathrm{DIF}}(q) \leq \frac{q}{p^{m-m'}-q}$
   - q is the sum of primitive and construction queries.

! For PGV-ELC: $\mathbf{Adv}_{\mathcal{C}}^{\mathrm{DIF}}(q) \leq \frac{q}{p^{n-n'}-q}$.

Modes security comparison

Mode	Primitive	COL	PRE	DIF
ELC-P	Perm(m)	q^2/p^ℓ	q/p^ℓ	$q/p^{m-m'}$
PGV-ELC	Block(κ, n)	q^2/p^ℓ	q/p^ℓ	$q/p^{n-n'}$
Sponge[1]	Perm(m)	$q^2/p^{\min\{\ell,m-m'\}}$	$q/p^{\min\{\ell,m-m'\}}$	$q^2/p^{m-m'}$

[1] DIF advantage drops to $q/p^{m'-m}$ for single-iteration.

Modes security comparison

Mode	Primitive	COL	PRE	DIF
ELC-P	Perm(m)	q^2/p^ℓ	q/p^ℓ	$q/p^{m-m'}$
PGV-ELC	Block(κ, n)	q^2/p^ℓ	q/p^ℓ	$q/p^{n-n'}$
Sponge[1]	Perm(m)	$q^2/p^{\min\{\ell, m-m'\}}$	$q/p^{\min\{\ell, m-m'\}}$	$q^2/p^{m-m'}$

Assuming $m' = n' + \kappa'$ and $m = n + \kappa$:

[1] DIF advantage drops to $q/p^{m'-m}$ for single-iteration.

Modes security comparison

Mode	Primitive	COL	PRE	DIF
ELC-P	Perm(m)	q^2/p^ℓ	q/p^ℓ	$q/p^{m-m'}$
PGV-ELC	Block(κ, n)	q^2/p^ℓ	q/p^ℓ	$q/p^{n-n'}$
Sponge[1]	Perm(m)	$q^2/p^{\min\{\ell,m-m'\}}$	$q/p^{\min\{\ell,m-m'\}}$	$q^2/p^{m-m'}$

Assuming $m' = n' + \kappa'$ and $m = n + \kappa$:

- ELC-P and PGV-LC have optimal COL and PRE resistance.

[1] DIF advantage drops to $q/p^{m'-m}$ for single-iteration.

Modes security comparison

Mode	Primitive	COL	PRE	DIF
ELC-P	Perm(m)	q^2/p^ℓ	q/p^ℓ	$q/p^{m-m'}$
PGV-ELC	Block(κ, n)	q^2/p^ℓ	q/p^ℓ	$q/p^{n-n'}$
Sponge[1]	Perm(m)	$q^2/p^{\min\{\ell,m-m'\}}$	$q/p^{\min\{\ell,m-m'\}}$	$q^2/p^{m-m'}$

Assuming $m' = n' + \kappa'$ and $m = n + \kappa$:

- ELC-P and PGV-LC have optimal COL and PRE resistance.
- COL and PRE resistance of Sponge are sub-optimal.

[1] DIF advantage drops to $q/p^{m'-m}$ for single-iteration.

Modes security comparison

Mode	Primitive	COL	PRE	DIF
ELC-P	Perm(m)	q^2/p^ℓ	q/p^ℓ	$q/p^{m-m'}$
PGV-ELC	Block(κ, n)	q^2/p^ℓ	q/p^ℓ	$q/p^{n-n'}$
Sponge[1]	Perm(m)	$q^2/p^{\min\{\ell,m-m'\}}$	$q/p^{\min\{\ell,m-m'\}}$	$q^2/p^{m-m'}$

Assuming $m' = n' + \kappa'$ and $m = n + \kappa$:

- ELC-P and PGV-LC have optimal COL and PRE resistance.
- COL and PRE resistance of Sponge are sub-optimal.
- ELC-P indifferentiability is better than PGV-ELC.

[1] DIF advantage drops to $q/p^{m'-m}$ for single-iteration.

Modes security comparison

Mode	Primitive	COL	PRE	DIF
ELC-P	Perm(m)	q^2/p^ℓ	q/p^ℓ	$q/p^{m-m'}$
PGV-ELC	Block(κ, n)	q^2/p^ℓ	q/p^ℓ	$q/p^{n-n'}$
Sponge[1]	Perm(m)	$q^2/p^{\min\{\ell,m-m'\}}$	$q/p^{\min\{\ell,m-m'\}}$	$q^2/p^{m-m'}$

Assuming $m' = n' + \kappa'$ and $m = n + \kappa$:

- ELC-P and PGV-LC have optimal COL and PRE resistance.
- COL and PRE resistance of Sponge are sub-optimal.
- ELC-P indifferentiability is better than PGV-ELC.

[1] DIF advantage drops to $q/p^{m'-m}$ for single-iteration.

Flexibility of the PGV-ELC/ELC-P **modes**

Arithmetization-Oriented *design strategies*:

Flexibility of the PGV-ELC/ELC-P modes

Arithmetization-Oriented *design strategies*:

- MARVELlous, Hades, `Anemoi`, GTDS, ... [3, 20, 11, 26]

Flexibility of the PGV-ELC/ELC-P modes

Arithmetization-Oriented *design strategies*:

- MARVELlous, Hades, `Anemoi`, GTDS, ... [3, 20, 11, 26]
- Arbitrarily sized block ciphers/permutations.

Flexibility of the PGV-ELC/ELC-P modes

Arithmetization-Oriented *design strategies*:

- MARVELlous, HADES, `Anemoi`, GTDS, ... [3, 20, 11, 26]
- Arbitrarily sized block ciphers/permutations.
- Secure parametrizations established via cryptanalysis.

Flexibility of the PGV-ELC/ELC-P modes

Arithmetization-Oriented *design strategies*:

- MARVELlous, Hades, `Anemoi`, GTDS, ... [3, 20, 11, 26]
- Arbitrarily sized block ciphers/permutations.
- Secure parametrizations established via cryptanalysis.
- Large instances more efficient than black-box combinations.

Flexibility of the PGV-ELC/ELC-P modes

Arithmetization-Oriented *design strategies*:

- MARVELlous, Hades, `Anemoi`, GTDS, … [3, 20, 11, 26]
- Arbitrarily sized block ciphers/permutations.
- Secure parametrizations established via cryptanalysis.
- Large instances more efficient than black-box combinations.
- ! Concrete instances subject to tailored attacks [4, 23].

Flexibility of the PGV-ELC/ELC-P modes

Arithmetization-Oriented *design strategies*:

- MARVELlous, Hades, Anemoi, GTDS, … [3, 20, 11, 26]
- Arbitrarily sized block ciphers/permutations.
- Secure parametrizations established via cryptanalysis.
- Large instances more efficient than black-box combinations.
- ! Concrete instances subject to tailored attacks [4, 23].

How to best use PGV-ELC/ELC-P modes?

Flexibility of the PGV-ELC/ELC-P modes

Arithmetization-Oriented *design strategies*:

- MARVELlous, Hades, Anemoi, GTDS, ... [3, 20, 11, 26]
- Arbitrarily sized block ciphers/permutations.
- Secure parametrizations established via cryptanalysis.
- Large instances more efficient than black-box combinations.
- ! Concrete instances subject to tailored attacks [4, 23].

How to best use PGV-ELC/ELC-P modes?

- High-arity/mixed-arity Merkle-Trees.

Flexibility of the PGV-ELC/ELC-P modes

Arithmetization-Oriented *design strategies*:

- MARVELlous, Hades, Anemoi, GTDS, … [3, 20, 11, 26]
- Arbitrarily sized block ciphers/permutations.
- Secure parametrizations established via cryptanalysis.
- Large instances more efficient than black-box combinations.
- ! Concrete instances subject to tailored attacks [4, 23].

How to best use PGV-ELC/ELC-P modes?

- High-arity/mixed-arity Merkle-Trees.
- Fiat-Shamir with known (reasonably short) message length.

Flexibility of the PGV-ELC/ELC-P modes

Arithmetization-Oriented *design strategies*:

- MARVELlous, Hades, Anemoi, GTDS, … [3, 20, 11, 26]
- Arbitrarily sized block ciphers/permutations.
- Secure parametrizations established via cryptanalysis.
- Large instances more efficient than black-box combinations.
- ! Concrete instances subject to tailored attacks [4, 23].

How to best use PGV-ELC/ELC-P modes?

- High-arity/mixed-arity Merkle-Trees.
- Fiat-Shamir with known (reasonably short) message length.
- ! Area cost, especially for HW implementations.

Flexibility of the PGV-ELC/ELC-P modes

Arithmetization-Oriented *design strategies*:

- MARVELlous, Hades, Anemoi, GTDS, … [3, 20, 11, 26]
- Arbitrarily sized block ciphers/permutations.
- Secure parametrizations established via cryptanalysis.
- Large instances more efficient than black-box combinations.
- ! Concrete instances subject to tailored attacks [4, 23].

How to best use PGV-ELC/ELC-P modes?

- High-arity/mixed-arity Merkle-Trees.
- Fiat-Shamir with known (reasonably short) message length.
- ! Area cost, especially for HW implementations.

Experiments

Target Design Strategies

HADES (POSEIDON):

- ‘Partial’ SPN structure.

Target Design Strategies

Hades (Poseidon):

- ‘Partial’ SPN structure.
- High number of rounds.

Target Design Strategies

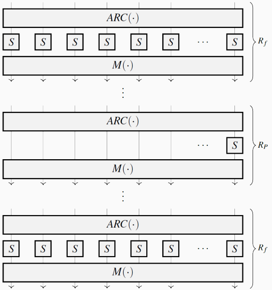
HADES (POSEIDON):

- 'Partial' SPN structure.
- High number of rounds.
- Lightweight key schedule.

Target Design Strategies

HADES (POSEIDON):

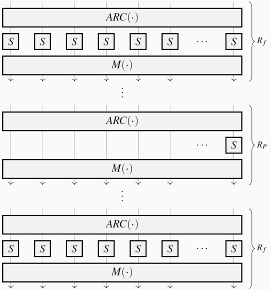
- 'Partial' SPN structure.
- High number of rounds.
- Lightweight key schedule.



Target Design Strategies

HADES (POSEIDON):

- 'Partial' SPN structure.
- High number of rounds.
- Lightweight key schedule.

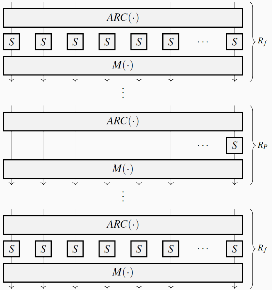


MARVELlous (*Rescue*):

Target Design Strategies

Hades (Poseidon):

- 'Partial' SPN structure.
- High number of rounds.
- Lightweight key schedule.



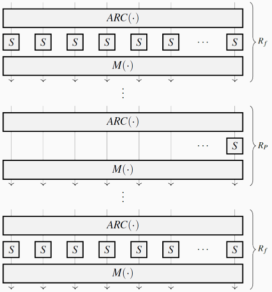
MARVELlous (*Rescue*):

- 'Double' SPN structure.

Target Design Strategies

HADES (POSEIDON):

- 'Partial' SPN structure.
- High number of rounds.
- Lightweight key schedule.



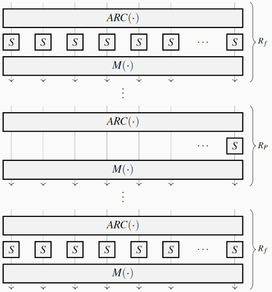
MARVELlous (*Rescue*):

- 'Double' SPN structure.
- Low number of rounds.

Target Design Strategies

HADES (POSEIDON):

- ‘Partial’ SPN structure.
- High number of rounds.
- Lightweight key schedule.



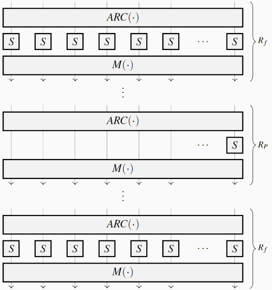
MARVELlous (*Rescue*):

- ‘Double’ SPN structure.
- Low number of rounds.
- Heavyweight key schedule.

Target Design Strategies

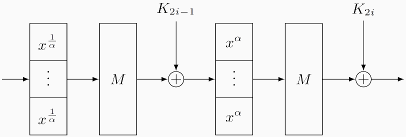
HADES (POSEIDON):

- 'Partial' SPN structure.
- High number of rounds.
- Lightweight key schedule.



MARVELlous (*Rescue*):

- 'Double' SPN structure.
- Low number of rounds.
- Heavyweight key schedule.



Concrete Instantiances

How did we parametrize HADES and *Rescue*:

Concrete Instantiances

How did we parametrize HADES and *Rescue*:

- Fields: Goldilocks (64 bits) and BLS12 scalar (256 bits).

Concrete Instantiances

How did we parametrize HADES and *Rescue*:

- Fields: Goldilocks (64 bits) and BLS12 scalar (256 bits).
- Sbox: $\alpha = \min\{a \mid \gcd(a, p-1) = 1\}$.

Concrete Instantiances

How did we parametrize HADES and *Rescue*:

- Fields: Goldilocks (64 bits) and BLS12 scalar (256 bits).
- Sbox: $\alpha = \min\{a \mid \gcd(a, p-1) = 1\}$.
- Affine layers use Hilbert's MDS matrix: $m_{i,j} = \frac{1}{i+j-1}$.
  - Also used for Poseidon's key scheduler.

Concrete Instantiances

How did we parametrize $\textsc{Hades}$ and *Rescue*:

- Fields: Goldilocks (64 bits) and BLS12 scalar (256 bits).
- Sbox: $\alpha = \min\{a \mid \gcd(a, p-1) = 1\}$.
- Affine layers use Hilbert's MDS matrix: $m_{i,j} = \frac{1}{i+j-1}$.
  - Also used for Poseidon's key scheduler.
- Round numbers computed according to [18, 27].

Concrete Instantiances

How did we parametrize $\textsc{Hades}$ and *Rescue*:

- Fields: Goldilocks (64 bits) and BLS12 scalar (256 bits).
- Sbox: $\alpha = \min\{a \mid \gcd(a, p-1) = 1\}$.
- Affine layers use Hilbert's MDS matrix: $m_{i,j} = \frac{1}{i+j-1}$.
    - Also used for Poseidon's key scheduler.
- Round numbers computed according to [18, 27].
- All compression/expansion matrices set to pseudo-identity.
    - Match the $\textsc{Trunc}$ mode used in e.g. $\textsc{Griffin}$ [16].

Concrete Instantiances

How did we parametrize $\textsc{Hades}$ and *Rescue*:

- Fields: Goldilocks (64 bits) and BLS12 scalar (256 bits).
- Sbox: $\alpha = \min\{a \mid \gcd(a, p-1) = 1\}$.
- Affine layers use Hilbert's MDS matrix: $m_{i,j} = \frac{1}{i+j-1}$.
    - ◇ Also used for Poseidon's key scheduler.
- Round numbers computed according to [18, 27].
- All compression/expansion matrices set to pseudo-identity.
    - ◇ Match the Trunc mode used in e.g. $\textsc{Griffin}$ [16].
- Security target: 128 bits of collision resistance.

Plain performance

Native execution performance:

[2] Intel Core i9-13900KF, Clear Linux, `libarith`, `icpx -O3 -march=native`

Plain performance

Native execution performance:

- C++ software implementation[2].

[2] Intel Core i9-13900KF, Clear Linux, `libarith`, `icpx -O3 -march=native`

Plain performance

Native execution performance:

- C++ software implementation[2].
- Light scheduler or long state $\Rightarrow$ PGV-ELC.

[2] Intel Core i9-13900KF, Clear Linux, `libarith`, `icpx -O3 -march=native`

Plain performance

Native execution performance:

- C++ software implementation[2].
- Light scheduler or long state $\Rightarrow$ PGV-ELC.
- Heavy scheduler and short state $\Rightarrow$ ELC-P.

[2] Intel Core i9-13900KF, Clear Linux, `libarith`, `icpx -O3 -march=native`

Plain performance

Native execution performance:

- C++ software implementation[2].
- Light scheduler or long state $\Rightarrow$ PGV-ELC.
- Heavy scheduler and short state $\Rightarrow$ ELC-P.
- ! PGV-ELC provides more parallelization opportunities.

[2] Intel Core i9-13900KF, Clear Linux, `libarith`, `icpx -O3 -march=native`

Plain performance

Native execution performance:

- C++ software implementation[2].
- Light scheduler or long state ⇒ PGV-ELC.
- Heavy scheduler and short state ⇒ ELC-P.
- ! PGV-ELC provides more parallelization opportunities.

		$\log_2(p) \approx 256$			$\log_2(p) \approx 64$		
	Rate	LC-P	PGV	Sponge	LC-P	PGV	Sponge
HADES	2:1	7.52 µs	12.3 µs	13.2 µs	4.12 µs	2.57 µs	8.49 µs
	4:1	19.3 µs	12.1 µs	28.2 µs	14.8 µs	7.02 µs	35.0 µs
	8:1	69.7 µs	36.8 µs	84.4 µs	164 µs	27.5 µs	223.6 µs
Rescue	2:1	183 µs	385 µs	208 µs	22.1 µs	24.2 µs	33.3 µs
	4:1	217 µs	401 µs	220 µs	47.1 µs	43.9 µs	58.9 µs
	8:1	320 µs	458 µs	354 µs	136 µs	92.4 µs	143 µs

[2] Intel Core i9-13900KF, Clear Linux, `libarith`, `icpx -O3 -march=native`

Groth16 benchmarks

We considered the Groth16 ZK-SNARK [21]:

[3] Same setup, + `libsnark`.

Groth16 benchmarks

We considered the Groth16 ZK-SNARK [21]:

- Requires a pairing-friendly elliptic curve like BLS12-381.

[3] Same setup, + `libsnark`.

Groth16 benchmarks

We considered the Groth16 ZK-SNARK [21]:

- Requires a pairing-friendly elliptic curve like BLS12-381.
- Preimage-verification circuit[3]:

[3] Same setup, + `libsnark`.

Groth16 benchmarks

We considered the Groth16 ZK-SNARK [21]:

- Requires a pairing-friendly elliptic curve like BLS12-381.
- Preimage-verification circuit[3]:
  - R1CS arithmetization: $\boldsymbol{Ax} \odot \boldsymbol{Bx} = \boldsymbol{Cx}$.

[3] Same setup, + `libsnark`.

Groth16 benchmarks

We considered the Groth16 ZK-SNARK [21]:

- Requires a pairing-friendly elliptic curve like BLS12-381.
- Preimage-verification circuit[3]:
  - R1CS arithmetization: $\boldsymbol{Ax} \odot \boldsymbol{Bx} = \boldsymbol{Cx}$.
  - Complexity depends mainly on the # of multiplications.

[3] Same setup, + `libsnark`.

Groth16 benchmarks

We considered the Groth16 ZK-SNARK [21]:

- Requires a pairing-friendly elliptic curve like BLS12-381.
- Preimage-verification circuit[3]:
  - R1CS arithmetization: $\boldsymbol{Ax} \odot \boldsymbol{Bx} = \boldsymbol{Cx}$.
  - Complexity depends mainly on the # of multiplications.

		# R1CS constraints			Proof Generation time		
	Ratio	LC-P	PGV	Sponge	LC-P	PGV	Sponge
Hades	2:1	221	221	246	72.9 ms	73.0 ms	75.8 ms
	4:1	268	218	293	83.0 ms	73.4 ms	89.4 ms
	8:1	368	268	393	105 ms	83.9 ms	115 ms
Rescue	2:1	240	432	252	67.2 ms	107 ms	67.7 ms
	4:1	264	480	270	71.1 ms	116 ms	73.4 ms
	8:1	384	528	432	102 ms	126 ms	110 ms

[3] Same setup, + `libsnark`.

Plonky2 **benchmarks**

We also considered the *Plonky2* ZK-SNARK [28]:

Plonky2 **benchmarks**

We also considered the *Plonky2* ZK-SNARK [28]:

- Uses FRI [5] over the Goldilocks field (high 2-adicity).

Plonky2 **benchmarks**

We also considered the *Plonky2* ZK-SNARK [28]:

- Uses FRI [5] over the Goldilocks field (high 2-adicity).
- Employs *Plonkish* arithmetization:

Plonky2 benchmarks

We also considered the *Plonky2* ZK-SNARK [28]:

- Uses FRI [5] over the Goldilocks field (high 2-adicity).
- Employs *Plonkish* arithmetization:
  - ◇ Based on $\mathcal{P}\mathfrak{lon}\mathcal{K}$ [14] $+$ custom gates.

Plonky2 **benchmarks**

We also considered the *Plonky2* ZK-SNARK [28]:

- Uses FRI [5] over the Goldilocks field (high 2-adicity).
- Employs *Plonkish* arithmetization:
  - ◇ Based on $\mathcal{P}\mathfrak{lon}\mathcal{K}$ [14] $+$ custom gates.
  - ◇ Applies optimizations to the circuit description.

Plonky2 benchmarks

We also considered the *Plonky2* ZK-SNARK [28]:

- Uses FRI [5] over the Goldilocks field (high 2-adicity).
- Employs *Plonkish* arithmetization:
  - Based on $\mathcal{P}\mathfrak{lon}\mathcal{K}$ [14] + custom gates.
  - Applies optimizations to the circuit description.

	Ratio	# gates LC-P	# gates PGV	# gates Sponge	Proof Generation time LC-P	Proof Generation time PGV	Proof Generation time Sponge
Hades	2:1	122	70	259	11.3 ms	11.1 ms	16.5 ms
	4:1	439	226	668	26.0 ms	16.2 ms	27.1 ms
	8:1	2065	847	2864	90.8 ms	47.5 ms	92.9 ms
Rescue	2:1	91	75	175	10.9 ms	8.58 ms	17.2 ms
	4:1	284	182	418	16.8 ms	11.5 ms	27.1 ms
	8:1	976	568	1213	47.9 ms	26.5 ms	49.0 ms

Merkle Tree arity benchmarks

Binary Merkle trees are the standard choice: larger arities?

Merkle Tree arity benchmarks

Binary Merkle trees are the standard choice: larger arities?

- We devised an optimized R1CS for MT openings [2]:

Merkle Tree arity benchmarks

Binary Merkle trees are the standard choice: larger arities?

- We devised an optimized R1CS for MT openings [2]:
  - Slight change in the opening structure (copath + full path).

Merkle Tree arity benchmarks

Binary Merkle trees are the standard choice: larger arities?

- We devised an optimized R1CS for MT openings [2]:
  - ◇ Slight change in the opening structure (copath $+$ full path).
  - ◇ up to 15% improvement for reasonable arities.

Merkle Tree arity benchmarks

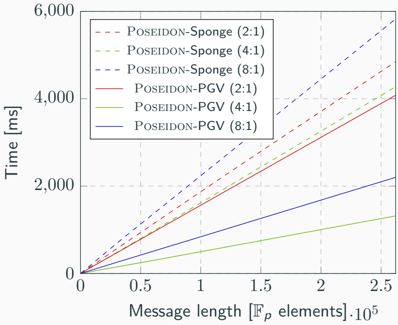
Binary Merkle trees are the standard choice: larger arities?

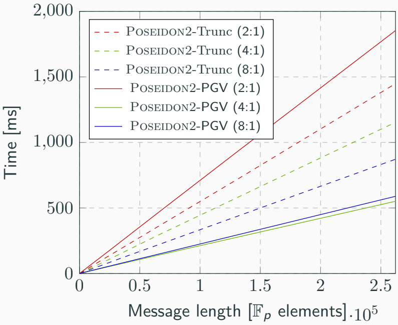
- We devised an optimized R1CS for MT openings [2]:
  - Slight change in the opening structure (copath $+$ full path).
  - up to 15% improvement for reasonable arities.
  - Scales with tree arity and compactness of the CF.

Merkle Tree arity benchmarks

Binary Merkle trees are the standard choice: larger arities?

- We devised an optimized R1CS for MT openings [2]:
  - ◇ Slight change in the opening structure (copath + full path).
  - ◇ up to 15% improvement for reasonable arities.
  - ◇ Scales with tree arity and compactness of the CF.





Roll credits

$\mathcal{T}he\ \ \mathcal{E}nd$

Thank you for your attention!

Any questions?

Bibliography i

Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen.
Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity.
In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pages 191–219, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

Elena Andreeva, Rishiraj Bhattacharyya, Arnab Roy, and Stefano Trevisani.
On Efficient and Secure Compression Functions for Arithmetization-Oriented Hashing.

In *2024 IEEE 37th Computer Security Foundations Symposium (CSF)*, pages 1–16, Los Alamitos, CA, USA, Jul 2024. IEEE Computer Society.

Tomer Ashur and Siemen Dhooghe.
Marvellous: a stark-friendly family of cryptographic primitives.
Cryptology ePrint Archive, Paper 2018/1098, 2018.
`https://eprint.iacr.org/2018/1098`.

Augustin Bariant, Aurélien Boeuf, Axel Lemoine, Irati Manterola Ayala, Morten Øygarden, Léo Perrin, and Håvard Raddum.
The algebraic freelunch efficient gröbner basis attacks against arithmetization-oriented primitives.
Cryptology ePrint Archive, Paper 2024/347, 2024.
https://eprint.iacr.org/2024/347.

Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev.
Fast Reed-Solomon Interactive Oracle Proofs of Proximity.
In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium*

on Automata, Languages, and Programming (ICALP 2018), volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche.
Sponge functions.
In *ECRYPT hash workshop*, volume 2007, 2007.

- Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche.
  On the indifferentiability of the sponge construction.
  In Nigel Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, pages 181–197, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- Amit Singh Bhati, Erik Pohle, Aysajan Abidin, Elena Andreeva, and Bart Preneel.
  Let's go eevee! a friendly and suitable family of aead modes for iot-to-cloud secure computation.
  In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, pages

2546–2560, New York, NY, USA, 2023. Association for Computing Machinery.

John Black, Phillip Rogaway, and Thomas Shrimpton.
Black-box analysis of the block-cipher-based hash-function constructions from pgv.
In Moti Yung, editor, *Advances in Cryptology — CRYPTO 2002*, pages 320–335, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

Joppe W. Bos and Peter L. Montgomery.
Montgomery arithmetic from a software perspective.
Cryptology ePrint Archive, Paper 2017/1057, 2017.
https://eprint.iacr.org/2017/1057.

Clémence Bouvier, Pierre Briaud, Pyrros Chaidos, Léo Perrin, Robin Salen, Vesselin Velichkov, and Danny Willems.
New design techniques for efficient arithmetization-oriented hash functions: `Anemoi` permutations and `Jive` compression mode.
In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 507–539, Cham, 2023. Springer Nature Switzerland.

Alessandro Chiesa, Dev Ojha, and Nicholas Spooner.
Fractal: Post-quantum and transparent recursive proofs from holography.
Cryptology ePrint Archive, Paper 2019/1076, 2019.
`https://eprint.iacr.org/2019/1076`.

Orr Dunkelman and Nathan Keller.
The effects of the omission of last round's mixcolumns on aes.
Information Processing Letters, 110(8):304–308, 2010.

Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru.
Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge.
Cryptology ePrint Archive, Paper 2019/953, 2019.
https://eprint.iacr.org/2019/953.

Shafi Goldwasser, Silvio Micali, and Charles Rackoff.
The knowledge complexity of interactive proof systems.
SIAM Journal on Computing, 18(1):186–208, 1989.

- Lorenzo Grassi, Yonglin Hao, Christian Rechberger, Markus Schofnegger, Roman Walch, and Qingju Wang.
  Horst meets fluid-spn: Griffin for zero-knowledge applications.
  Cryptology ePrint Archive, Paper 2022/403, 2022.
  https://eprint.iacr.org/2022/403.
- Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, Markus Schofnegger, and Roman Walch.
  Hash functions monolith for zk applications: May the speed of sha-3 be with you.
  Cryptology ePrint Archive, Paper 2023/1025, 2023.
  https://eprint.iacr.org/2023/1025.

Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger.
Poseidon: A new hash function for Zero-Knowledge proof systems.
In *30th USENIX Security Symposium (USENIX Security 21)*, pages 519–535. USENIX Association, aug 2021.

Lorenzo Grassi, Dmitry Khovratovich, and Markus Schofnegger.
Poseidon2: A faster version of the poseidon hash function.
Cryptology ePrint Archive, Paper 2023/323, 2023.
`https://eprint.iacr.org/2023/323`.

Lorenzo Grassi, Reinhard Lüftenegger, Christian Rechberger, Dragos Rotaru, and Markus Schofnegger.
On a generalization of substitution-permutation networks: The hades design strategy.
Cryptology ePrint Archive, Paper 2019/1107, 2019.
https://eprint.iacr.org/2019/1107.

Jens Groth.
On the size of pairing-based non-interactive arguments.
In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

Dmitry Khovratovich, Mario Marhuenda Beltrán, and Bart Mennink.
Generic security of the safe api and its applications.
In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023*, pages 301–327, Singapore, 2023. Springer Nature Singapore.

Katharina Koschatko, Reinhard Lüftenegger, and Christian Rechberger.
Exploring the six worlds of gröbner basis cryptanalysis: Application to anemoi.
IACR Transactions on Symmetric Cryptology, 2024(4):138–190, Dec. 2024.

Ralph Charles Merkle.
Secrecy, Authentication, and Public Key Systems.
PhD thesis, Stanford University, Stanford, CA, USA, 1979.
AAI8001972.

Bart Preneel, René Govaerts, and Joos Vandewalle.
Hash functions based on block ciphers: A synthetic approach.
In *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 368–378. Springer, 1993.

Arnab Roy and Matthias Johann Steiner.
Generalized triangular dynamical system: An algebraic system for constructing cryptographic permutations over finite fields.
Cryptology ePrint Archive, Paper 2024/1316, 2024.

Alan Szepieniec, Tomer Ashur, and Siemen Dhooghe.
Rescue-prime: a standard specification (sok).
Cryptology ePrint Archive, Paper 2020/1143, 2020.
https://eprint.iacr.org/2020/1143.

Polygon Zero Team.
Plonky2: Fast recursive arguments with plonk and fri, September 2022.
https://github.com/0xPolygonZero/plonky2/blob/main/plonky2/plonky2.pdf.