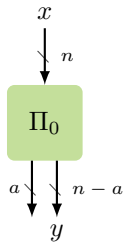
Short-Input Random Oracle from Public Random Permutations

Hrithik Nandi

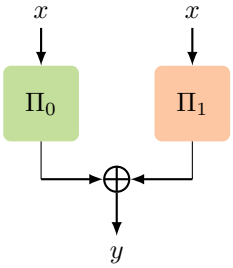
Institute for Advancing Intelligence (IAI), TCG CREST
&
Ramakrishna Mission Vivekananda Educational and Research Institute

March 20, 2026

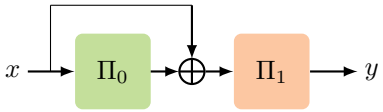
- Π_0 and Π_1 are two *secret* (*keyed*) permutations.



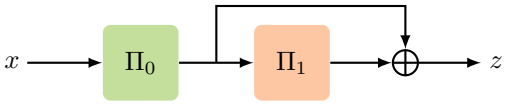
Truncation



Sum Of Permutations



EDM



EDMD

Keyed setting

Secret Permutations

- Construction based on *secret* (*keyed*) permutations.
- Adversary only accesses the **construction**.

$\Longrightarrow$ **Indistinguishability**

Keyless setting

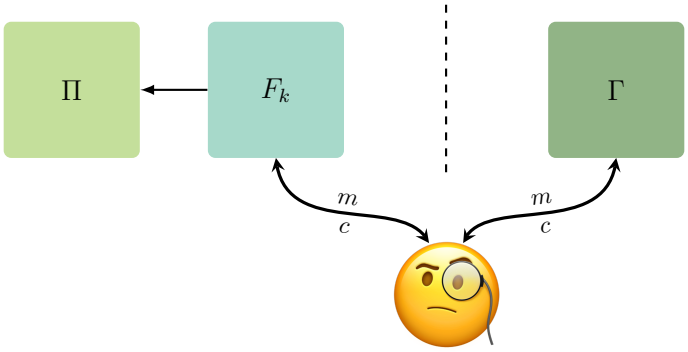
Public Permutations

- Construction based on *public* permutations.
- Adversary accesses the **construction** and the **underlying permutations**.

$\Longrightarrow$ **Indifferentiability**

Security Model: Indistinguishability

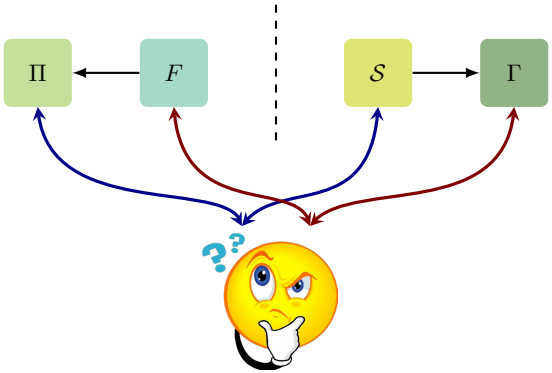
- F_k is a keyed function based on secret permutation Π, and Γ is a random function.



$$\mathsf{Adv}^{\mathrm{PRF}}_{\mathcal{A},\mathsf{F}}(q) \coloneqq |\Pr[k \xleftarrow{\$} \mathcal{K} \colon \mathcal{A}^{\mathsf{F}_k(\cdot)} \to 1] - \Pr[\mathsf{f} \xleftarrow{\$} \mathrm{Func}(n) \colon \mathcal{A}^{\mathsf{f}(\cdot)} \to 1]|$$

Security Model: Indifferentiability

- F is a function based on public permutation Π and Γ is a random function.
- $\mathcal{S}$ is a simulator, which has access to Γ and imitates the public permutation Π.



$$\mathrm{Adv}^{\mathrm{indiff}}_{F^{\Pi},\Gamma^{\mathcal{S}}}(\mathcal{A}) := |\Pr[\mathcal{A}^{F,\Pi} \to 1] - \Pr[\mathcal{A}^{\Gamma,\mathcal{S}} \to 1]|$$

Indifferentiability of The Class of Functions based on Two Permutations

- Let Π_0, Π_1 be two permutations over $\{0,1\}^n$.
- Let $\alpha_i, \beta_i, \gamma_i \in \mathrm{GL}(\mathbf{F}_2^n) \cup \{0\}$ and $x \in \{0,1\}^n$:

$$\mathtt{F}^{\Pi_0,\Pi_1}(x) = \gamma_1(x) \oplus \gamma_2(\Pi_0(\alpha_1 x)) \oplus \gamma_3(\Pi_1(\beta_1 x \oplus \beta_2 \Pi_0(\alpha_1 x))) .$$

- $\mathtt{F}$ can be viewed by the triple (A, Π_0, Π_1) where

$$A = \begin{pmatrix} \alpha_1 & 0 & 0 \\ \beta_1 & \beta_2 & 0 \\ 0 & \gamma_2 & \gamma_3 \end{pmatrix}.$$

We assume $\gamma_1 = 0$, since we can always consider the modified function $\mathtt{F}'(x) = \mathtt{F}(x) \oplus \gamma_1 x$.

Observations: Trivial Attacks in Some Cases

$$\mathtt{F}^{\Pi_0,\Pi_1}(x) = \gamma_2(\Pi_0(\alpha_1 x)) \oplus \gamma_3(\Pi_1(\beta_1 x \oplus \beta_2 \Pi_0(\alpha_1 x)))$$

$$A = \begin{pmatrix} \alpha_1 & 0 & 0 \\ \beta_1 & \beta_2 & 0 \\ 0 & \gamma_2 & \gamma_3 \end{pmatrix}$$

Observations: Trivial Attacks in Some Cases

$$\mathtt{F}^{\Pi_0,\Pi_1}(x) = \gamma_2(\Pi_0(\alpha_1 x)) \oplus \gamma_3(\Pi_1(\beta_1 x \oplus \beta_2 \Pi_0(\alpha_1 x)))$$

$$A = \begin{pmatrix} \alpha_1 & 0 & 0 \\ \beta_1 & \beta_2 & 0 \\ 0 & \gamma_2 & \gamma_3 \end{pmatrix}$$

- A contains either a zero row or a zero column: either the function F becomes constant, or the influence of one permutation is lost. Easily attacked with at most 1 construction query and 1 primitive query.

Observations: Trivial Attacks in Some Cases

$$\mathtt{F}^{\Pi_0,\Pi_1}(x) = \gamma_2(\Pi_0(\alpha_1 x)) \oplus \gamma_3(\Pi_1(\beta_1 x \oplus \beta_2 \Pi_0(\alpha_1 x)))$$

$$A = \begin{pmatrix} \alpha_1 & 0 & 0 \\ \beta_1 & \beta_2 & 0 \\ 0 & \gamma_2 & \gamma_3 \end{pmatrix}$$

- A contains either a zero row or a zero column: either the function $\mathtt{F}$ becomes constant, or the influence of one permutation is lost. Easily attacked with at most 1 construction query and 1 primitive query.
- A is such that $\beta_1 = \gamma_2 = 0$: $\mathtt{F}(x) = \gamma_3(\Pi_1(\beta_2 \Pi_0(\alpha_1 x)))$.
  - Choose an element x and $z \leftarrow \mathtt{C}(\alpha_1^{-1} x)$.
  - $v \leftarrow \mathtt{P}_1^{-1}(\gamma_3^{-1} z)$.
  - $u \leftarrow \mathtt{P}_0^{-1}(\beta_2^{-1} v)$.
  - Check whether the output equals x.

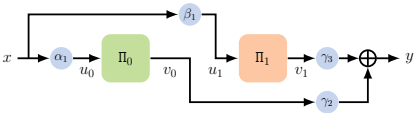
  Attacked with at most 1 construction query and 2 primitive queries.

Remaining Possible Constructions

$$\mathrm{F}^{\Pi_0,\Pi_1}(x) = \gamma_2(\Pi_0(\alpha_1 x)) \oplus \gamma_3(\Pi_1(\beta_1 x \oplus \beta_2 \Pi_0(\alpha_1 x)))$$

Remaining Possible Constructions

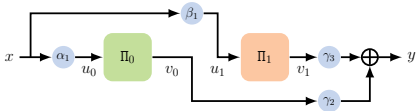
$$\mathtt{F}^{\Pi_0,\Pi_1}(x) = \gamma_2(\Pi_0(\alpha_1 x)) \oplus \gamma_3(\Pi_1(\beta_1 x \oplus \beta_2 \Pi_0(\alpha_1 x)))$$



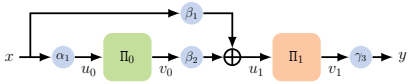
$A_{\mathtt{SOP}}$ $(\beta_2 = 0 \wedge \beta_1, \gamma_2 \neq 0)$

Remaining Possible Constructions

$$\mathtt{F}^{\Pi_0,\Pi_1}(x) = \gamma_2(\Pi_0(\alpha_1 x)) \oplus \gamma_3(\Pi_1(\beta_1 x \oplus \beta_2 \Pi_0(\alpha_1 x)))$$



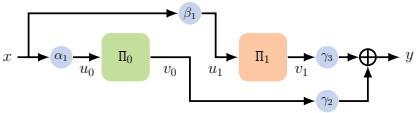
$A_{\mathtt{SOP}}$ $(\beta_2 = 0 \wedge \beta_1, \gamma_2 \neq 0)$



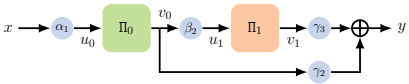
$A_{\mathtt{EDM}}$ $(\gamma_2 = 0 \wedge \beta_1, \beta_2 \neq 0)$

Remaining Possible Constructions

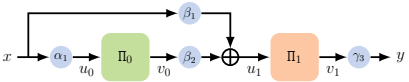
$$\mathtt{F}^{\Pi_0,\Pi_1}(x) = \gamma_2(\Pi_0(\alpha_1 x)) \oplus \gamma_3(\Pi_1(\beta_1 x \oplus \beta_2 \Pi_0(\alpha_1 x)))$$



$A_{\texttt{SOP}}$ $(\beta_2 = 0 \wedge \beta_1, \gamma_2 \neq 0)$



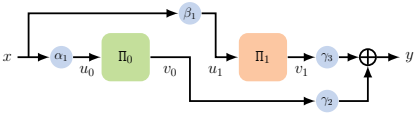
$A_{\texttt{EDMD}}$ $(\beta_1 = 0 \wedge \beta_2, \gamma_2 \neq 0)$



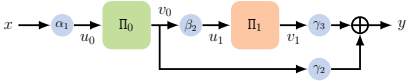
$A_{\texttt{EDM}}$ $(\gamma_2 = 0 \wedge \beta_1, \beta_2 \neq 0)$

Remaining Possible Constructions

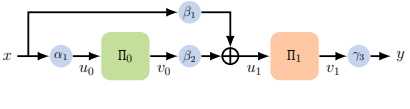
$$\mathtt{F}^{\Pi_0,\Pi_1}(x) = \gamma_2(\Pi_0(\alpha_1 x)) \oplus \gamma_3(\Pi_1(\beta_1 x \oplus \beta_2 \Pi_0(\alpha_1 x)))$$



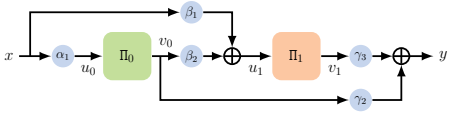
$A_{\texttt{SOP}}$ $(\beta_2 = 0 \wedge \beta_1, \gamma_2 \neq 0)$



$A_{\texttt{EDMD}}$ $(\beta_1 = 0 \wedge \beta_2, \gamma_2 \neq 0)$



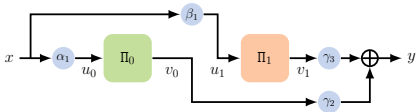
$A_{\texttt{EDM}}$ $(\gamma_2 = 0 \wedge \beta_1, \beta_2 \neq 0)$



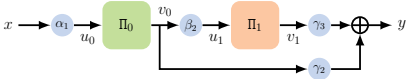
$A_{\texttt{DMDM}}$ $(\beta_1, \beta_2, \gamma_2 \neq 0)$

Remaining Possible Constructions

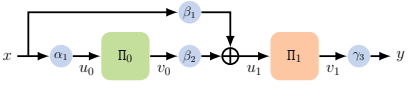
$$\mathtt{F}^{\Pi_0,\Pi_1}(x) = \gamma_2(\Pi_0(\alpha_1 x)) \oplus \gamma_3(\Pi_1(\beta_1 x \oplus \beta_2 \Pi_0(\alpha_1 x)))$$



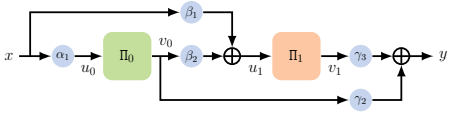
$A_{\mathtt{SOP}}$ $(\beta_2 = 0 \wedge \beta_1, \gamma_2 \neq 0)$



$A_{\mathtt{EDMD}}$ $(\beta_1 = 0 \wedge \beta_2, \gamma_2 \neq 0)$



$A_{\mathtt{EDM}}$ $(\gamma_2 = 0 \wedge \beta_1, \beta_2 \neq 0)$



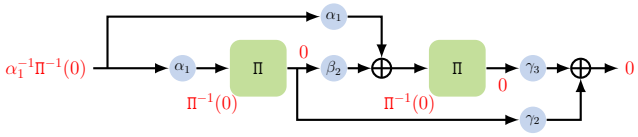
$A_{\mathtt{DMDM}}$ $(\beta_1, \beta_2, \gamma_2 \neq 0)$

- We analyze these four constructions under three different assumptions on Π_0 and Π_1.

Π_0 is Identical to Π_1: Trivial Attack

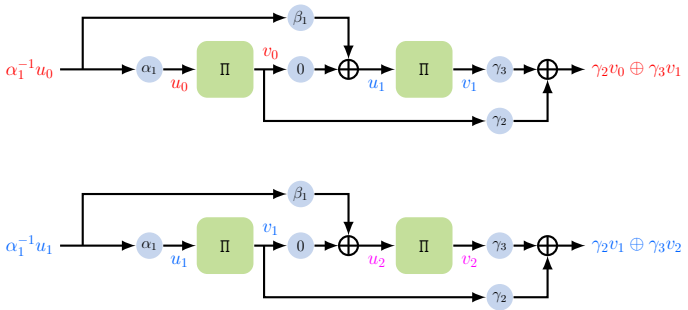
$$\mathtt{F}(x) = \gamma_2(\Pi(\alpha_1 x)) \oplus \gamma_3(\Pi(\beta_1 x \oplus \beta_2 \Pi(\alpha_1 x)))$$

- If $\alpha_1 = \beta_1$ then we must have $\mathtt{F}(\alpha_1^{-1}\Pi^{-1}(0)) = 0$. This gives a indifferentiability attack.



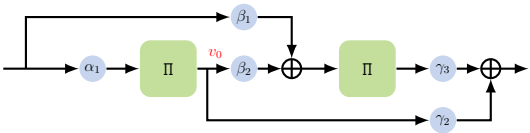
Π_0 is Identical to Π_1: Non-trivial Attack

- Let $\beta_2 = 0$ and $\beta_1 \alpha_1^{-1}$ has order ℓ.
- Choose u_0 and compute $u_i = \beta_1 \alpha_1^{-1} u_{i-1}$ for $i = 1, \ldots, \ell$.



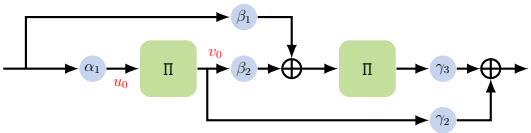
- Consider the equations $\gamma_2 v_i \oplus \gamma_3 v_{i+1} = y_i$ for all i mod ℓ.
- If the system has full rank, compute $v_0, \ldots, v_{\ell-1}$ and check whether $\mathsf{P}^{-1}(v_0) = u_0$.
- Otherwise, there exist $a_0, \ldots, a_{\ell-1}$, not all zero, such that $\sum_{i=0}^{\ell-1} a_i y_i = 0$.

Π_0 is Identical to Π_1: Creating a Chain



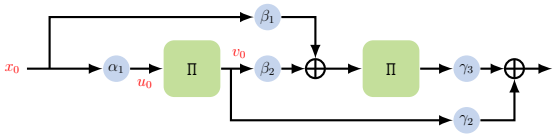
- Choose an element v_0.

Π_0 is Identical to Π_1: Creating a Chain



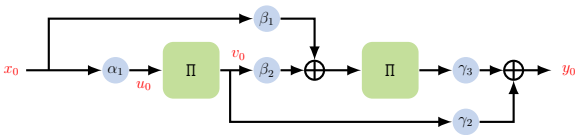
- Choose an element v_0.
- Make a primitive query: $u_0 \leftarrow \mathtt{P}^{-1}(v_0)$.

Π_0 is Identical to Π_1: Creating a Chain



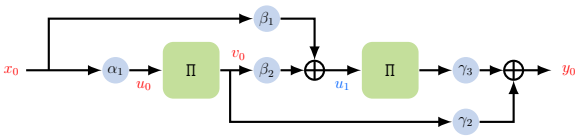
- Choose an element v_0.
- Make a primitive query: $u_0 \leftarrow \mathtt{P}^{-1}(v_0)$.
- Construct: $x_0 \leftarrow \alpha_1^{-1} u_0$.

Π_0 is Identical to Π_1: Creating a Chain



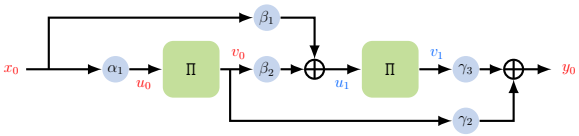
- Choose an element v_0.
- Make a primitive query: $u_0 \leftarrow \mathtt{P}^{-1}(v_0)$.
- Construct: $x_0 \leftarrow \alpha_1^{-1} u_0$.
- Make a construction query: $y_0 \leftarrow \mathtt{C}(x_0)$.

Π_0 is Identical to Π_1: Creating a Chain



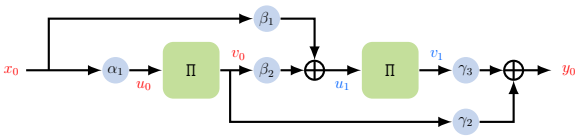
- Choose an element v_0.
- Make a primitive query: $u_0 \leftarrow \mathtt{P}^{-1}(v_0)$.
- Construct: $x_0 \leftarrow \alpha_1^{-1} u_0$.
- Make a construction query: $y_0 \leftarrow \mathtt{C}(x_0)$.
- $u_1 \leftarrow \beta_1 \alpha_1^{-1} u_0 \oplus \beta_2 v_0$.

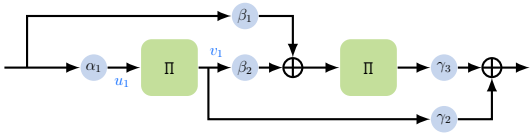
Π_0 is Identical to Π_1: Creating a Chain



- Choose an element v_0.
- Make a primitive query: $u_0 \leftarrow \mathtt{P}^{-1}(v_0)$.
- Construct: $x_0 \leftarrow \alpha_1^{-1} u_0$.
- Make a construction query: $y_0 \leftarrow \mathtt{C}(x_0)$.
- $u_1 \leftarrow \beta_1 \alpha_1^{-1} u_0 \oplus \beta_2 v_0$.
- $v_1 \leftarrow \gamma_3^{-1}(y_0 \oplus \gamma_2 v_0)$.

Π_0 is Identical to Π_1: Creating a Chain

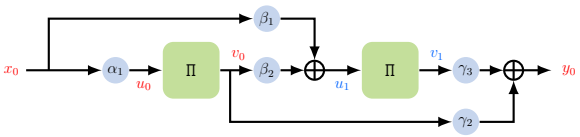




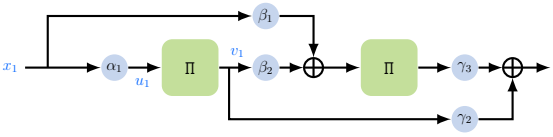
- Choose an element v_0.
- Make a primitive query: $u_0 \leftarrow \mathtt{P}^{-1}(v_0)$.
- Construct: $x_0 \leftarrow \alpha_1^{-1} u_0$.
- Make a construction query: $y_0 \leftarrow \mathtt{C}(x_0)$.
- $u_1 \leftarrow \beta_1 \alpha_1^{-1} u_0 \oplus \beta_2 v_0$.
- $v_1 \leftarrow \gamma_3^{-1}(y_0 \oplus \gamma_2 v_0)$.

- We already have v_1 and u_1.

Π_0 is Identical to Π_1: Creating a Chain

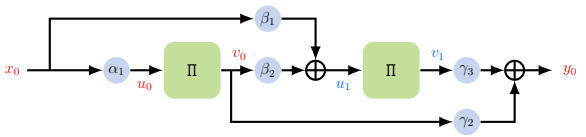


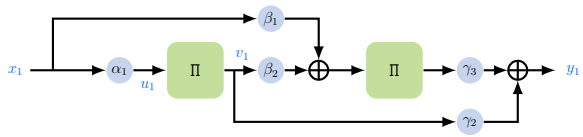
- Choose an element v_0.
- Make a primitive query: $u_0 \leftarrow \mathtt{P}^{-1}(v_0)$.
- Construct: $x_0 \leftarrow \alpha_1^{-1} u_0$.
- Make a construction query: $y_0 \leftarrow \mathtt{C}(x_0)$.
- $u_1 \leftarrow \beta_1 \alpha_1^{-1} u_0 \oplus \beta_2 v_0$.
- $v_1 \leftarrow \gamma_3^{-1}(y_0 \oplus \gamma_2 v_0)$.



- We already have v_1 and u_1.
- Construct: $x_1 \leftarrow \alpha_1^{-1} u_1$.

Π_0 is Identical to Π_1: Creating a Chain

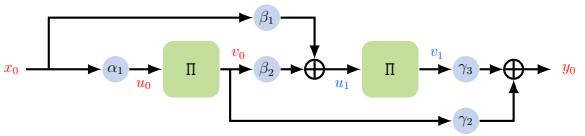


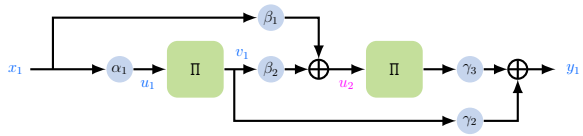


- Choose an element v_0.
- Make a primitive query: $u_0 \leftarrow \mathtt{P}^{-1}(v_0)$.
- Construct: $x_0 \leftarrow \alpha_1^{-1} u_0$.
- Make a construction query: $y_0 \leftarrow \mathtt{C}(x_0)$.
- $u_1 \leftarrow \beta_1 \alpha_1^{-1} u_0 \oplus \beta_2 v_0$.
- $v_1 \leftarrow \gamma_3^{-1}(y_0 \oplus \gamma_2 v_0)$.

- We already have v_1 and u_1.
- Construct: $x_1 \leftarrow \alpha_1^{-1} u_1$.
- Make a construction query: $y_1 \leftarrow \mathtt{C}(x_1)$.

Π_0 is Identical to Π_1: Creating a Chain

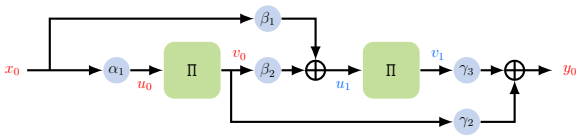


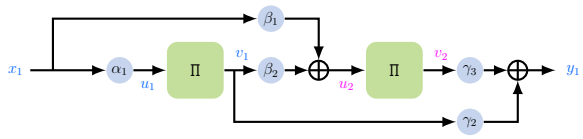


- Choose an element v_0.
- Make a primitive query: $u_0 \leftarrow \mathtt{P}^{-1}(v_0)$.
- Construct: $x_0 \leftarrow \alpha_1^{-1} u_0$.
- Make a construction query: $y_0 \leftarrow \mathtt{C}(x_0)$.
- $u_1 \leftarrow \beta_1 \alpha_1^{-1} u_0 \oplus \beta_2 v_0$.
- $v_1 \leftarrow \gamma_3^{-1}(y_0 \oplus \gamma_2 v_0)$.

- We already have v_1 and u_1.
- Construct: $x_1 \leftarrow \alpha_1^{-1} u_1$.
- Make a construction query: $y_1 \leftarrow \mathtt{C}(x_1)$.
- $u_2 \leftarrow \beta_1 \alpha_1^{-1} u_1 \oplus \beta_2 v_1$.

Π_0 is Identical to Π_1: Creating a Chain





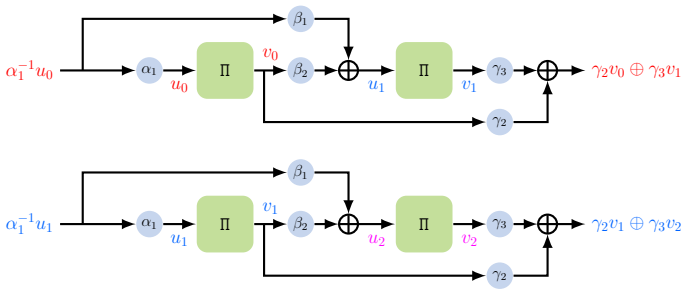
- Choose an element v_0.
- Make a primitive query: $u_0 \leftarrow \mathtt{P}^{-1}(v_0)$.
- Construct: $x_0 \leftarrow \alpha_1^{-1} u_0$.
- Make a construction query: $y_0 \leftarrow \mathtt{C}(x_0)$.
- $u_1 \leftarrow \beta_1 \alpha_1^{-1} u_0 \oplus \beta_2 v_0$.
- $v_1 \leftarrow \gamma_3^{-1}(y_0 \oplus \gamma_2 v_0)$.

- We already have v_1 and u_1.
- Construct: $x_1 \leftarrow \alpha_1^{-1} u_1$.
- Make a construction query: $y_1 \leftarrow \mathtt{C}(x_1)$.
- $u_2 \leftarrow \beta_1 \alpha_1^{-1} u_1 \oplus \beta_2 v_1$.
- $v_2 \leftarrow \gamma_3^{-1}(y_1 \oplus \gamma_2 v_1)$.

Definittion & Property of a Chain

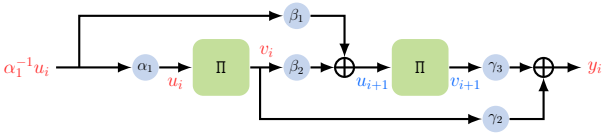
- $((u_0, v_0), (u_1, v_1), (u_2, v_2) \ldots, (u_k, v_k))$ is said to be a *k-chain* iff

$$v_i = \mathtt{P}(u_i) \text{ and } u_{i+1} = \beta_1 \alpha_1^{-1} u_i \oplus \beta_2 v_i, i \in [k-1].$$



- In the real world, $\mathtt{F}(\alpha_1^{-1}u_0) \oplus \cdots \oplus \mathtt{F}(\alpha_1^{-1}u_i) = \gamma_2 v_0 \oplus \bigoplus_{j=1}^{i}(\gamma_2 \oplus \gamma_3)v_j \oplus \gamma_3 v_{i+1}$, however for any efficient simulator it is hard to maintain such an equivalency, even for moderately large k.

Π_0 is Identical to Π_1: The Chaining Attack



function DISTINGUISHER($\mathcal{D}$)
  Fix a constant v_0;
  $\mathcal{C} \leftarrow \texttt{CHAIN}(v_0, q)$;
  if $\mathcal{C} = (0, \star)$ **then**
    return 0
  View $\mathcal{C}$ as $(1, (u_0, v_0), \ldots, (u_q, v_q))$;
  $u \leftarrow \texttt{P}^{-1}(v_q)$;
  if $u = u_q$ **then**
    return 1
  else
    return 0

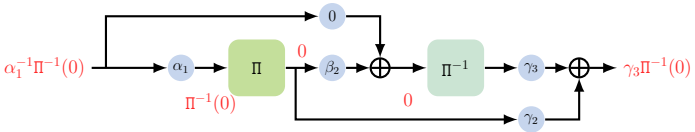
Candidate	Parameters	Indifferentiability
$A^{\Pi}_{\texttt{SOP}}$, $A^{\Pi}_{\texttt{EDMD}}$ $A^{\Pi}_{\texttt{EDM}}$, $A^{\Pi}_{\texttt{DMDM}}$	$\beta_2 = 0$ and $\left((\beta_1\alpha_1^{-1})^\ell = 1 \wedge \ell \text{ is small}\right)$	$1 - O\left(q_{\mathcal{S}}/2^n\right)$
$A^{\Pi}_{\texttt{SOP}}$, $A^{\Pi}_{\texttt{EDMD}}$ $A^{\Pi}_{\texttt{EDM}}$, $A^{\Pi}_{\texttt{DMDM}}$	$\beta_2 \neq 0$ or $\left((\beta_1\alpha_1^{-1})^\ell = 1 \wedge \ell \text{ is large}\right)$	$1 - 1/c - cq_{\mathcal{S}}/2^n$

Table: Summary of indifferentiability advantage bounds. $q_{\mathcal{S}}$ denotes the number of queries simulator makes on expectation and $c > 1$ is a constant.

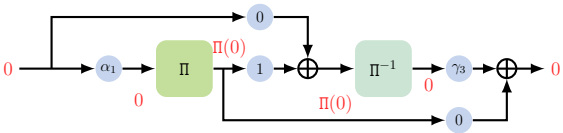
Π_1 is Inverse of Π_0: Trivial Attack

$$\mathtt{F}(x) = \gamma_2(\Pi(\alpha_1(x))) \oplus \gamma_3(\Pi^{-1}(\beta_1 x \oplus \beta_2 \Pi(\alpha_1(x))))$$

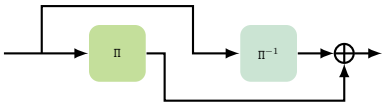
- If $\beta_1 = 0$, then we must have $\mathtt{F}(\alpha_1^{-1}\Pi^{-1}(0)) = \gamma_3\Pi^{-1}(0)$. This gives a indifferentiability attack.



- If $\beta_2 = 1$ and $\gamma_2 = 0$, then $\mathtt{F}$ has a fixed-point at 0. This gives a indifferentiability attack.



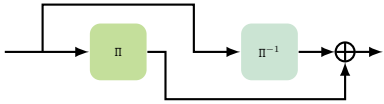
Result on SOPI construction



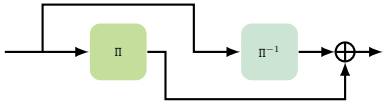
- Dodis et al. proposed [DPP08, Lemma 4] a simulator for `SOPI`.
- They proved a birthday-bound (strong) indifferentiability security with the proposed simulator.

• [DPP08] Dodis, Y., Pietrzak, K., Puniya, P.: A New Mode of Operation for Block Ciphers and Length-Preserving MACs. EUROCRYPT 2008.

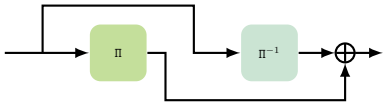
SOPI: The Simulator & The Attack



SOPI: The Simulator & The Attack



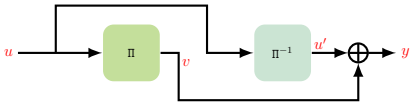
The Simulator



The Simulator

- It maintains a list $\mathcal{L}$ of all previous query-response pairs (u, v).

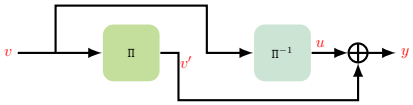
SOPI: The Simulator & The Attack



The Simulator

- It maintains a list $\mathcal{L}$ of all previous query-response pairs (u, v).
- Forward query u: it searches $\mathcal{L}$ for a pair of the form (u', u). Then it queries the random function Γ to find the output $\Gamma(u)$. It records the pair (u, v) in $\mathcal{L}$, where $v = u' \oplus \Gamma(u)$, and returns v.

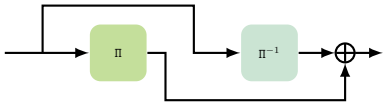
SOPI: The Simulator & The Attack



The Simulator

- It maintains a list $\mathcal{L}$ of all previous query-response pairs (u, v).
- Forward query u: it searches $\mathcal{L}$ for a pair of the form (u', u). Then it queries the random function Γ to find the output $\Gamma(u)$. It records the pair (u, v) in $\mathcal{L}$, where $v = u' \oplus \Gamma(u)$, and returns v.
- Backward query v: it searches $\mathcal{L}$ for a pair (v, v'). If it finds, then it queries the random function to find the output $\Gamma(v)$. It records the pair (u, v) in $\mathcal{L}$, where $u = v' \oplus \Gamma(v)$, and returns u.

SOPI: The Simulator & The Attack

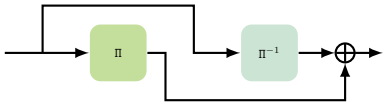


The Simulator

- It maintains a list $\mathcal{L}$ of all previous query-response pairs (u, v).
- Forward query u: it searches $\mathcal{L}$ for a pair of the form (u', u). Then it queries the random function Γ to find the output $\Gamma(u)$. It records the pair (u, v) in $\mathcal{L}$, where $v = u' \oplus \Gamma(u)$, and returns v.
- Backward query v: it searches $\mathcal{L}$ for a pair (v, v'). If it finds, then it queries the random function to find the output $\Gamma(v)$. It records the pair (u, v) in $\mathcal{L}$, where $u = v' \oplus \Gamma(v)$, and returns u.

Our Attack to this simulator

SOPI: The Simulator & The Attack



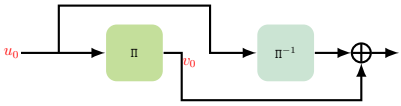
The Simulator

- It maintains a list $\mathcal{L}$ of all previous query-response pairs (u, v).
- Forward query u: it searches $\mathcal{L}$ for a pair of the form (u', u). Then it queries the random function Γ to find the output $\Gamma(u)$. It records the pair (u, v) in $\mathcal{L}$, where $v = u' \oplus \Gamma(u)$, and returns v.
- Backward query v: it searches $\mathcal{L}$ for a pair (v, v'). If it finds, then it queries the random function to find the output $\Gamma(v)$. It records the pair (u, v) in $\mathcal{L}$, where $u = v' \oplus \Gamma(v)$, and returns u.

Our Attack to this simulator

- Sample $u_0 \leftarrow \{0, 1^n\}$.

SOPI: The Simulator & The Attack



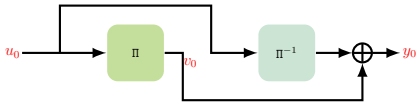
The Simulator

- It maintains a list $\mathcal{L}$ of all previous query-response pairs (u, v).
- Forward query u: it searches $\mathcal{L}$ for a pair of the form (u', u). Then it queries the random function Γ to find the output $\Gamma(u)$. It records the pair (u, v) in $\mathcal{L}$, where $v = u' \oplus \Gamma(u)$, and returns v.
- Backward query v: it searches $\mathcal{L}$ for a pair (v, v'). If it finds, then it queries the random function to find the output $\Gamma(v)$. It records the pair (u, v) in $\mathcal{L}$, where $u = v' \oplus \Gamma(v)$, and returns u.

Our Attack to this simulator

- Sample $u_0 \leftarrow \{0, 1^n\}$.
- Query $\mathtt{P}$ with u_0 and let the response be v_0.

SOPI: The Simulator & The Attack



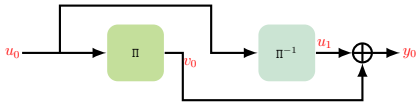
The Simulator

- It maintains a list $\mathcal{L}$ of all previous query-response pairs (u, v).
- Forward query u: it searches $\mathcal{L}$ for a pair of the form (u', u). Then it queries the random function Γ to find the output $\Gamma(u)$. It records the pair (u, v) in $\mathcal{L}$, where $v = u' \oplus \Gamma(u)$, and returns v.
- Backward query v: it searches $\mathcal{L}$ for a pair (v, v'). If it finds, then it queries the random function to find the output $\Gamma(v)$. It records the pair (u, v) in $\mathcal{L}$, where $u = v' \oplus \Gamma(v)$, and returns u.

Our Attack to this simulator

- Sample $u_0 \leftarrow \{0, 1^n\}$.
- Query $\mathtt{P}$ with u_0 and let the response be v_0.
- Query $\mathtt{C}$ with u_0 and let the response be y_0.

SOPI: The Simulator & The Attack



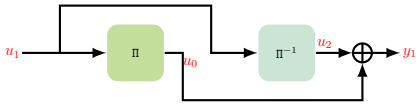
The Simulator

- It maintains a list $\mathcal{L}$ of all previous query-response pairs (u, v).
- Forward query u: it searches $\mathcal{L}$ for a pair of the form (u', u). Then it queries the random function Γ to find the output $\Gamma(u)$. It records the pair (u, v) in $\mathcal{L}$, where $v = u' \oplus \Gamma(u)$, and returns v.
- Backward query v: it searches $\mathcal{L}$ for a pair (v, v'). If it finds, then it queries the random function to find the output $\Gamma(v)$. It records the pair (u, v) in $\mathcal{L}$, where $u = v' \oplus \Gamma(v)$, and returns u.

Our Attack to this simulator

- Sample $u_0 \leftarrow \{0, 1^n\}$.
- Query `P` with u_0 and let the response be v_0.
- Query `C` with u_0 and let the response be y_0.
- Set $u_1 = y_0 \oplus v_0$, and query `C` with u_1 and let the response be y_1.

SOPI: The Simulator & The Attack



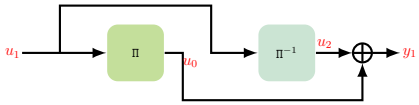
The Simulator

- It maintains a list $\mathcal{L}$ of all previous query-response pairs (u, v).
- Forward query u: it searches $\mathcal{L}$ for a pair of the form (u', u). Then it queries the random function Γ to find the output $\Gamma(u)$. It records the pair (u, v) in $\mathcal{L}$, where $v = u' \oplus \Gamma(u)$, and returns v.
- Backward query v: it searches $\mathcal{L}$ for a pair (v, v'). If it finds, then it queries the random function to find the output $\Gamma(v)$. It records the pair (u, v) in $\mathcal{L}$, where $u = v' \oplus \Gamma(v)$, and returns u.

Our Attack to this simulator

- Sample $u_0 \leftarrow \{0, 1^n\}$.
- Query `P` with u_0 and let the response be v_0.
- Query `C` with u_0 and let the response be y_0.
- Set $u_1 = y_0 \oplus v_0$, and query `C` with u_1 and let the response be y_1.
- Set $u_2 = y_1 \oplus u_0$, and query `P` with u_2 and let the response be v.

SOPI: The Simulator & The Attack



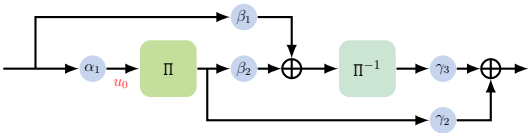
The Simulator

- It maintains a list $\mathcal{L}$ of all previous query-response pairs (u, v).
- Forward query u: it searches $\mathcal{L}$ for a pair of the form (u', u). Then it queries the random function Γ to find the output $\Gamma(u)$. It records the pair (u, v) in $\mathcal{L}$, where $v = u' \oplus \Gamma(u)$, and returns v.
- Backward query v: it searches $\mathcal{L}$ for a pair (v, v'). If it finds, then it queries the random function to find the output $\Gamma(v)$. It records the pair (u, v) in $\mathcal{L}$, where $u = v' \oplus \Gamma(v)$, and returns u.

Our Attack to this simulator

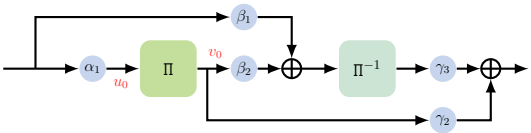
- Sample $u_0 \leftarrow \{0, 1^n\}$.
- Query `P` with u_0 and let the response be v_0.
- Query `C` with u_0 and let the response be y_0.
- Set $u_1 = y_0 \oplus v_0$, and query `C` with u_1 and let the response be y_1.
- Set $u_2 = y_1 \oplus u_0$, and query `P` with u_2 and let the response be v.
- If $v = u_1$ then return 0, else return 1.

Π_1 is Inverse of Π_0: Creating another Chain



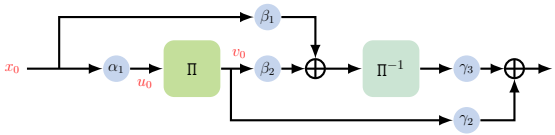
- Choose an element u_0.

Π_1 is Inverse of Π_0: Creating another Chain



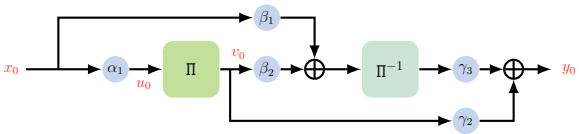
- Choose an element u_0.
- Make a primitive query: $v_0 \leftarrow \mathsf{P}(u_0)$.

Π_1 is Inverse of Π_0: Creating another Chain



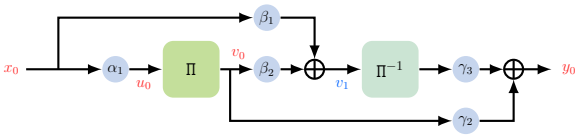
- Choose an element u_0.
- Make a primitive query: $v_0 \leftarrow \mathsf{P}(u_0)$.
- Construct: $x_0 \leftarrow \alpha_1^{-1} u_0$.

Π_1 is Inverse of Π_0: Creating another Chain



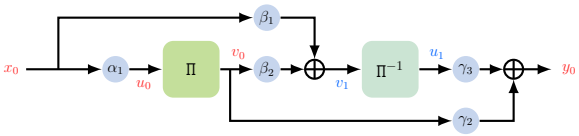
- Choose an element u_0.
- Make a primitive query: $v_0 \leftarrow \mathtt{P}(u_0)$.
- Construct: $x_0 \leftarrow \alpha_1^{-1} u_0$.
- Make a construction query: $y_0 \leftarrow \mathtt{C}(x_0)$.

Π_1 is Inverse of Π_0: Creating another Chain



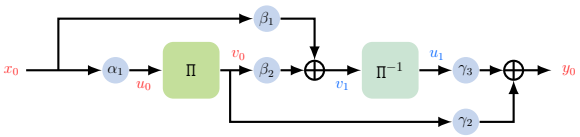
- Choose an element u_0.
- Make a primitive query: $v_0 \leftarrow \mathtt{P}(u_0)$.
- Construct: $x_0 \leftarrow \alpha_1^{-1} u_0$.
- Make a construction query: $y_0 \leftarrow \mathtt{C}(x_0)$.
- $v_1 \leftarrow \beta_1 \alpha_1^{-1} u_0 \oplus \beta_2 v_0$.

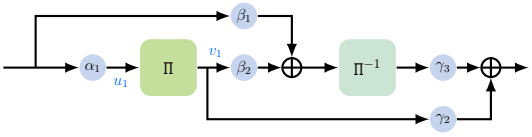
Π_1 is Inverse of Π_0: Creating another Chain



- Choose an element u_0.
- Make a primitive query: $v_0 \leftarrow \mathsf{P}(u_0)$.
- Construct: $x_0 \leftarrow \alpha_1^{-1} u_0$.
- Make a construction query: $y_0 \leftarrow \mathsf{C}(x_0)$.
- $v_1 \leftarrow \beta_1 \alpha_1^{-1} u_0 \oplus \beta_2 v_0$.
- $u_1 \leftarrow \gamma_3^{-1}(y_0 \oplus \gamma_2 v_0)$.

Π_1 is Inverse of Π_0: Creating another Chain

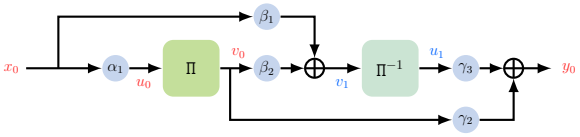




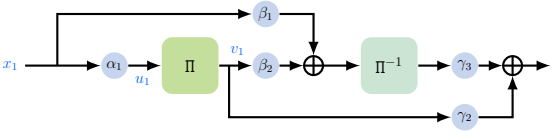
- Choose an element u_0.
- Make a primitive query: $v_0 \leftarrow \mathtt{P}(u_0)$.
- Construct: $x_0 \leftarrow \alpha_1^{-1} u_0$.
- Make a construction query: $y_0 \leftarrow \mathtt{C}(x_0)$.
- $v_1 \leftarrow \beta_1 \alpha_1^{-1} u_0 \oplus \beta_2 v_0$.
- $u_1 \leftarrow \gamma_3^{-1}(y_0 \oplus \gamma_2 v_0)$.

- We already have u_1 and v_1.

Π_1 is Inverse of Π_0: Creating another Chain

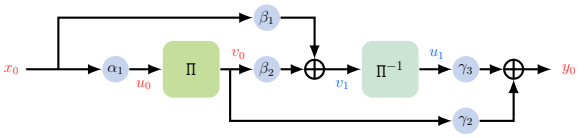


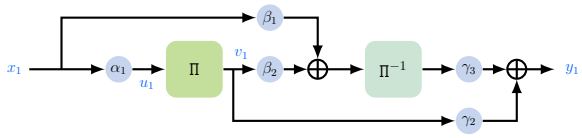
- Choose an element u_0.
- Make a primitive query: $v_0 \leftarrow \mathtt{P}(u_0)$.
- Construct: $x_0 \leftarrow \alpha_1^{-1} u_0$.
- Make a construction query: $y_0 \leftarrow \mathtt{C}(x_0)$.
- $v_1 \leftarrow \beta_1 \alpha_1^{-1} u_0 \oplus \beta_2 v_0$.
- $u_1 \leftarrow \gamma_3^{-1}(y_0 \oplus \gamma_2 v_0)$.



- We already have u_1 and v_1.
- Construct: $x_1 \leftarrow \alpha_1^{-1} u_1$.

Π_1 is Inverse of Π_0: Creating another Chain

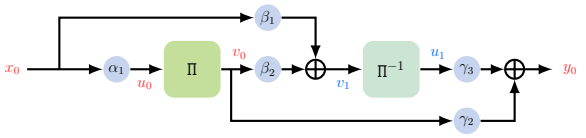


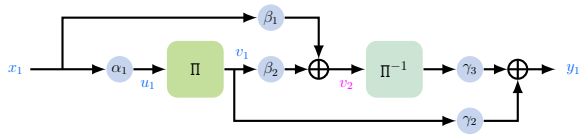


- Choose an element u_0.
- Make a primitive query: $v_0 \leftarrow \mathtt{P}(u_0)$.
- Construct: $x_0 \leftarrow \alpha_1^{-1} u_0$.
- Make a construction query: $y_0 \leftarrow \mathtt{C}(x_0)$.
- $v_1 \leftarrow \beta_1 \alpha_1^{-1} u_0 \oplus \beta_2 v_0$.
- $u_1 \leftarrow \gamma_3^{-1}(y_0 \oplus \gamma_2 v_0)$.

- We already have u_1 and v_1.
- Construct: $x_1 \leftarrow \alpha_1^{-1} u_1$.
- Make a construction query: $y_1 \leftarrow \mathtt{C}(x_1)$.

Π_1 is Inverse of Π_0: Creating another Chain

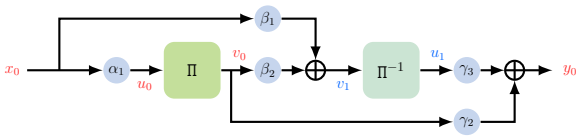


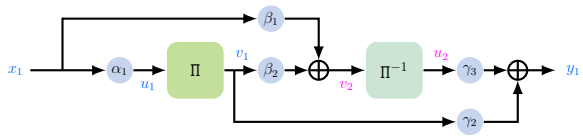


- Choose an element u_0.
- Make a primitive query: $v_0 \leftarrow \mathtt{P}(u_0)$.
- Construct: $x_0 \leftarrow \alpha_1^{-1} u_0$.
- Make a construction query: $y_0 \leftarrow \mathtt{C}(x_0)$.
- $v_1 \leftarrow \beta_1 \alpha_1^{-1} u_0 \oplus \beta_2 v_0$.
- $u_1 \leftarrow \gamma_3^{-1}(y_0 \oplus \gamma_2 v_0)$.

- We already have u_1 and v_1.
- Construct: $x_1 \leftarrow \alpha_1^{-1} u_1$.
- Make a construction query: $y_1 \leftarrow \mathtt{C}(x_1)$.
- $v_2 \leftarrow \beta_1 \alpha_1^{-1} u_1 \oplus \beta_2 v_1$.

Π_1 is Inverse of Π_0: Creating another Chain





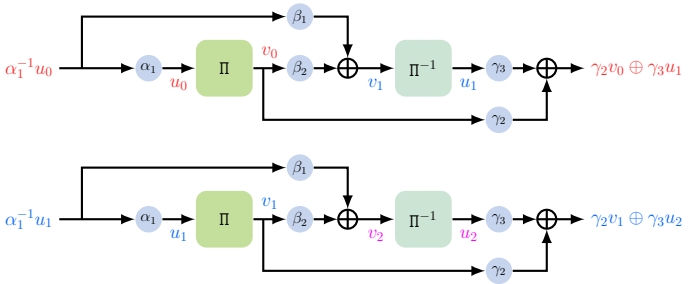
- Choose an element u_0.
- Make a primitive query: $v_0 \leftarrow \mathtt{P}(u_0)$.
- Construct: $x_0 \leftarrow \alpha_1^{-1} u_0$.
- Make a construction query: $y_0 \leftarrow \mathtt{C}(x_0)$.
- $v_1 \leftarrow \beta_1 \alpha_1^{-1} u_0 \oplus \beta_2 v_0$.
- $u_1 \leftarrow \gamma_3^{-1}(y_0 \oplus \gamma_2 v_0)$.

- We already have u_1 and v_1.
- Construct: $x_1 \leftarrow \alpha_1^{-1} u_1$.
- Make a construction query: $y_1 \leftarrow \mathtt{C}(x_1)$.
- $v_2 \leftarrow \beta_1 \alpha_1^{-1} u_1 \oplus \beta_2 v_1$.
- $u_2 \leftarrow \gamma_3^{-1}(y_1 \oplus \gamma_2 v_1)$.

Π_1 is Inverse of Π_0: The Chaining Attack

- $((u_0, v_0), (u_1, v_1), (u_2, v_2) \ldots, (u_k, v_k))$ is said to be another k-*chain* iff

$$v_i = \mathtt{P}(u_i) \text{ and } u_{i+1} = \gamma_3^{-1}(\mathtt{C}(\alpha^{-1}u_i) \oplus \gamma_2 v_i), i \in [k-1].$$



- In the real world, $\mathtt{F}(\alpha_1^{-1}u_0) \oplus \cdots \oplus \mathtt{F}(\alpha_1^{-1}u_i) = \bigoplus_{j=0}^{i} \gamma_2 v_j \oplus \bigoplus_{j=0}^{i} \gamma_3 u_{j+1}$, however for any efficient simulator it is hard to maintain such an equivalency, even for moderately large k.
- The goal of the distinguisher is to create a k-chain.

Π_1 is Inverse of Π_0: Summary of Our Results

Candidate	Parameters	Indifferentiability
$A^{\Pi\pm}_{\texttt{SOP}}$, $A^{\Pi\pm}_{\texttt{EDMD}}$ $A^{\Pi\pm}_{\texttt{EDM}}$, $A^{\Pi\pm}_{\texttt{DMDM}}$	$\beta_1 = 0$ or $(\beta_2, \gamma_2) = (1, 0)$	$1 - O\big(q_{\mathcal{S}}/2^n\big)$
$A^{\Pi\pm}_{\texttt{SOP}}$, $A^{\Pi\pm}_{\texttt{EDMD}}$ $A^{\Pi\pm}_{\texttt{EDM}}$, $A^{\Pi\pm}_{\texttt{DMDM}}$	$(\beta_1, \beta_2, \gamma_2) \neq (0, 1, 0)$	$1 - 1/c - cq_{\mathcal{S}}/2^n$

Table: Summary of indifferentiability advantage bounds. $q_{\mathcal{S}}$ denotes the number of queries simulator makes on expectation and $c > 1$ is a constant.

Π_0 and Π_1 are Independent: Our Results

Candidate	Parameters	Indifferentiability
$A^{\Pi_0,\Pi_1}_{\texttt{SOP}}$, $A^{\Pi_0,\Pi_1}_{\texttt{EDMD}}$	$\star$	$O\left(\sqrt{q^3/2^{2n}}\right)$
$A^{\Pi_0,\Pi_1}_{\texttt{EDM}}$, $A^{\Pi_0,\Pi_1}_{\texttt{DMDM}}$	$\star$	$\Theta\left(q^2/2^n\right)$

Table: Summary of indifferentiability advantage bounds for $q \leq 2^{n-3}$ adversarial queries. The logarithmic terms are suppressed in the bounds.

Indifferentiability of Short-Input Extendable-Output Random Oracle

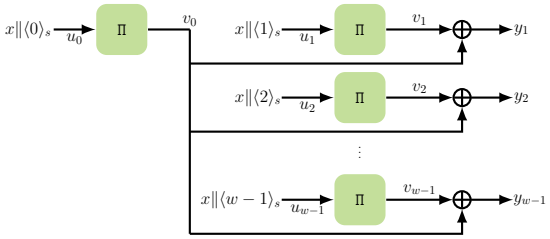


Figure: The $\texttt{XORP}^{\Pi}_{w}$ construction.

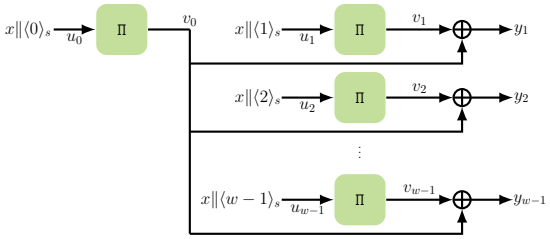


Figure: The $\texttt{XORP}^{\Pi}_{w}$ construction.

Forward query with u

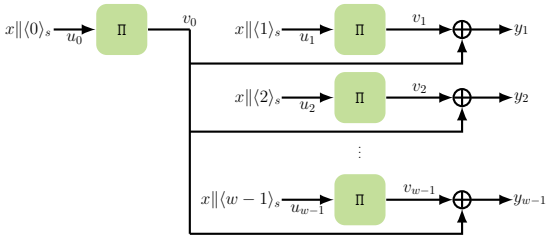


Figure: The $\mathtt{XORP}^{\Pi}_{w}$ construction.

Forward query with u

- Extract x from u: $x\|\langle r\rangle_s \leftarrow u$.

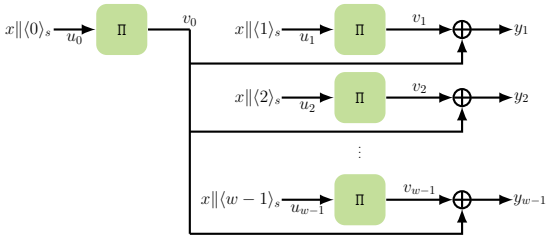


Figure: The $\mathtt{XORP}^{\Pi}_{w}$ construction.

Forward query with u

- Extract x from u: $x\|\langle r\rangle_s \leftarrow u$.
- $(y_1\| \cdots \|y_{w-1}) \leftarrow \Gamma(x)$.

Indifferentiability of $\mathtt{XORP}^{\Pi}_{w}$

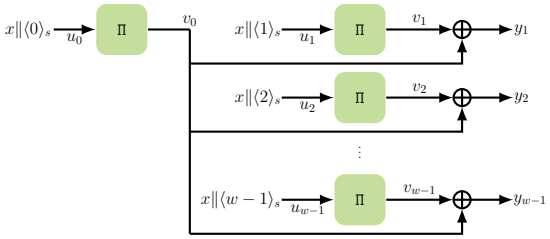


Figure: The $\mathtt{XORP}^{\Pi}_{w}$ construction.

Forward query with u

- Extract x from u: $x\|\langle r\rangle_s \leftarrow u$.
- $(y_1\| \cdots \|y_{w-1}) \leftarrow \Gamma(x)$.
- Check if $y_i = 0 \vee y_i = y_j$.

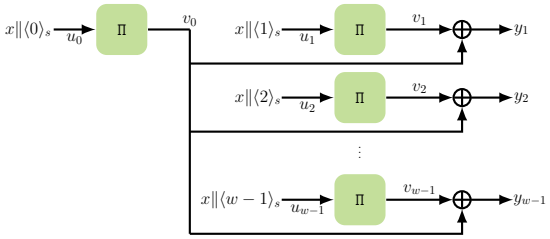


Figure: The $\mathtt{XORP}^{\Pi}_{w}$ construction.

Forward query with u

- Extract x from u: $x\|\langle r\rangle_s \leftarrow u$.
- $(y_1\| \cdots \|y_{w-1}) \leftarrow \Gamma(x)$.
- Check if $y_i = 0 \vee y_i = y_j$.
- Choose v_0 consistently and store $(u_j, v_0 \oplus y_j)$.

Indifferentiability of $\mathtt{XORP}^{\Pi}_{w}$

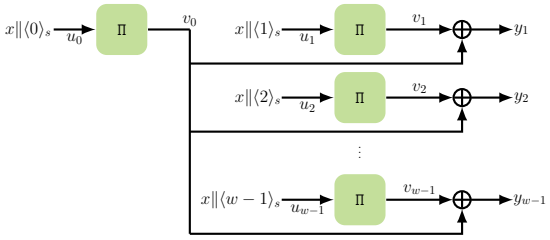


Figure: The $\mathtt{XORP}^{\Pi}_{w}$ construction.

Forward query with u

- Extract x from u: $x\|\langle r\rangle_s \leftarrow u$.
- $(y_1\| \cdots \|y_{w-1}) \leftarrow \Gamma(x)$.
- Check if $y_i = 0 \vee y_i = y_j$.
- Choose v_0 consistently and store $(u_j, v_0 \oplus y_j)$.

Inverse query with v

Indifferentiability of $\mathtt{XORP}^{\Pi}_{w}$

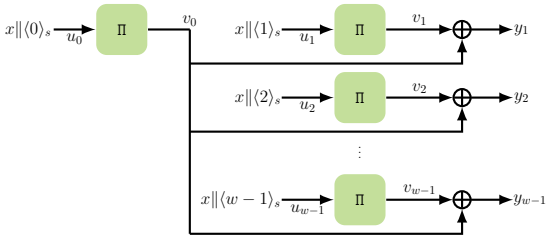


Figure: The $\mathtt{XORP}^{\Pi}_{w}$ construction.

Forward query with u

- Extract x from u: $x\|\langle r\rangle_s \leftarrow u$.
- $(y_1\| \cdots \|y_{w-1}) \leftarrow \Gamma(x)$.
- Check if $y_i = 0 \vee y_i = y_j$.
- Choose v_0 consistently and store $(u_j, v_0 \oplus y_j)$.

Inverse query with v

- Choose u_0 consistently and $x\|\langle r\rangle_s \leftarrow u$.

Indifferentiability of $\mathtt{XORP}^{\Pi}_{w}$

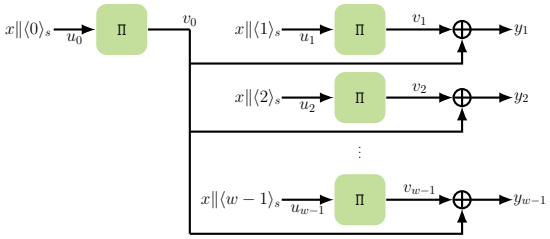


Figure: The $\mathtt{XORP}^{\Pi}_{w}$ construction.

Forward query with u

- Extract x from u: $x\|\langle r\rangle_s \leftarrow u$.
- $(y_1\| \cdots \|y_{w-1}) \leftarrow \Gamma(x)$.
- Check if $y_i = 0 \vee y_i = y_j$.
- Choose v_0 consistently and store $(u_j, v_0 \oplus y_j)$.

Inverse query with v

- Choose u_0 consistently and $x\|\langle r\rangle_s \leftarrow u$.
- $(y_1\| \cdots \|y_{w-1}) \leftarrow \Gamma(x)$.

Indifferentiability of $\mathtt{XORP}^{\Pi}_{w}$

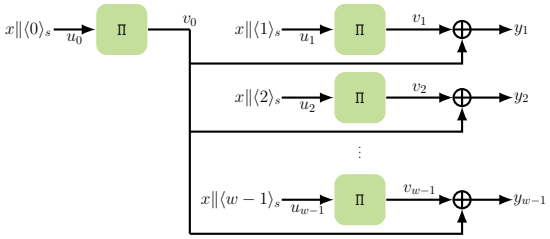


Figure: The $\mathtt{XORP}^{\Pi}_{w}$ construction.

Forward query with u

- Extract x from u: $x\|\langle r\rangle_s \leftarrow u$.
- $(y_1\| \cdots \|y_{w-1}) \leftarrow \Gamma(x)$.
- Check if $y_i = 0 \vee y_i = y_j$.
- Choose v_0 consistently and store $(u_j, v_0 \oplus y_j)$.

Inverse query with v

- Choose u_0 consistently and $x\|\langle r\rangle_s \leftarrow u$.
- $(y_1\| \cdots \|y_{w-1}) \leftarrow \Gamma(x)$.
- Check if $y_i = 0 \vee y_i = y_j$.

Indifferentiability of $\mathtt{XORP}^{\Pi}_{w}$

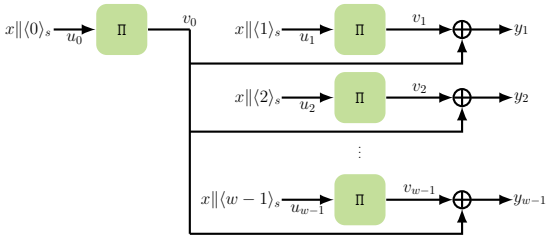


Figure: The $\mathtt{XORP}^{\Pi}_{w}$ construction.

Forward query with u

- Extract x from u: $x\|\langle r\rangle_s \leftarrow u$.
- $(y_1\| \cdots \|y_{w-1}) \leftarrow \Gamma(x)$.
- Check if $y_i = 0 \vee y_i = y_j$.
- Choose v_0 consistently and store $(u_j, v_0 \oplus y_j)$.

Inverse query with v

- Choose u_0 consistently and $x\|\langle r\rangle_s \leftarrow u$.
- $(y_1\| \cdots \|y_{w-1}) \leftarrow \Gamma(x)$.
- Check if $y_i = 0 \vee y_i = y_j$.
- Set $v_j \leftarrow y_j \oplus y_r \oplus v$.

Indifferentiability of $\mathtt{XORP}^{\Pi}_{w}$

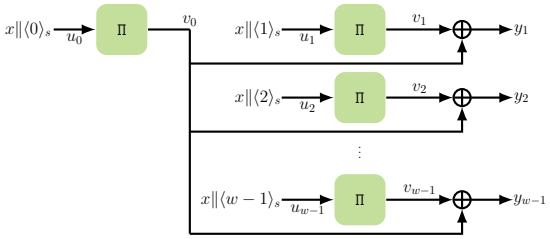


Figure: The $\mathtt{XORP}^{\Pi}_{w}$ construction.

Forward query with u

- Extract x from u: $x\|\langle r\rangle_s \leftarrow u$.
- $(y_1\| \cdots \|y_{w-1}) \leftarrow \Gamma(x)$.
- Check if $y_i = 0 \vee y_i = y_j$.
- Choose v_0 consistently and store $(u_j, v_0 \oplus y_j)$.

Inverse query with v

- Choose u_0 consistently and $x\|\langle r\rangle_s \leftarrow u$.
- $(y_1\| \cdots \|y_{w-1}) \leftarrow \Gamma(x)$.
- Check if $y_i = 0 \vee y_i = y_j$.
- Set $v_j \leftarrow y_j \oplus y_r \oplus v$.
- If all the v_j values are consistent then store (u_j, v_j).

Indifferentiability of $\mathtt{XORP}^{\Pi}_{w}$

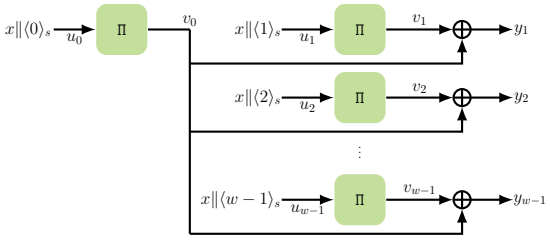


Figure: The $\mathtt{XORP}^{\Pi}_{w}$ construction.

Forward query with u

- Extract x from u: $x\|\langle r\rangle_s \leftarrow u$.
- $(y_1\| \cdots \|y_{w-1}) \leftarrow \Gamma(x)$.
- Check if $y_i = 0 \vee y_i = y_j$.
- Choose v_0 consistently and store $(u_j, v_0 \oplus y_j)$.

Inverse query with v

- Choose u_0 consistently and $x\|\langle r\rangle_s \leftarrow u$.
- $(y_1\| \cdots \|y_{w-1}) \leftarrow \Gamma(x)$.
- Check if $y_i = 0 \vee y_i = y_j$.
- Set $v_j \leftarrow y_j \oplus y_r \oplus v$.
- If all the v_j values are consistent then store (u_j, v_j).
- Otherwise, make another attempt with a new u_0.

Indifferentiability of $\mathtt{XORP}^{\Pi}_{w}$

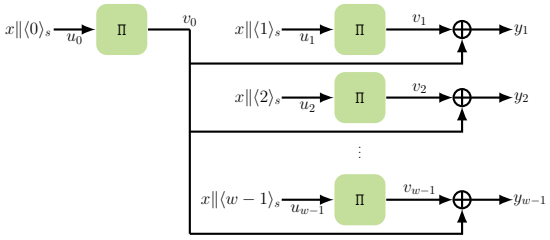


Figure: The $\mathtt{XORP}^{\Pi}_{w}$ construction.

Forward query with u

- Extract x from u: $x\|\langle r\rangle_s \leftarrow u$.
- $(y_1\| \cdots \|y_{w-1}) \leftarrow \Gamma(x)$.
- Check if $y_i = 0 \vee y_i = y_j$.
- Choose v_0 consistently and store $(u_j, v_0 \oplus y_j)$.

Inverse query with v

- Choose u_0 consistently and $x\|\langle r\rangle_s \leftarrow u$.
- $(y_1\| \cdots \|y_{w-1}) \leftarrow \Gamma(x)$.
- Check if $y_i = 0 \vee y_i = y_j$.
- Set $v_j \leftarrow y_j \oplus y_r \oplus v$.
- If all the v_j values are consistent then store (u_j, v_j).
- Otherwise, make another attempt with a new u_0.

$\mathtt{XORP}^{\Pi}_{w}$ is indifferentiable up to $2^{2n/3}$ queries.

For More Details








Accepted at EUROCRYPT 2026

https://ia.cr/2026/336

Thank You!

Questions?