Radbound University

Revisiting the Indifferentiability of the Sum of Permutations

Aldo Gunsing, Ritam Bhaumik, Ashwin Jha, Bart Mennink, Yaobin Shen

Crypto 2023

PRP versus PRF

- Many symmetric cryptographic schemes are based on pseudorandom permutations (PRPs) like AES

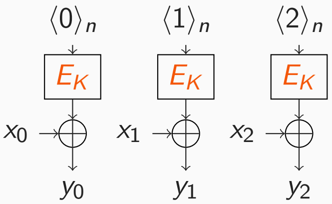
PRP versus PRF

- Many symmetric cryptographic schemes are based on pseudorandom permutations (PRPs) like AES
- A lot of modes only use the forward direction, not making use of the invertibility

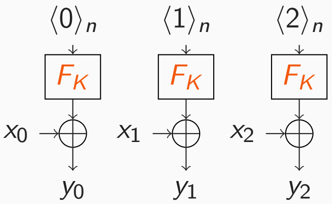
PRP versus PRF

- Many symmetric cryptographic schemes are based on pseudorandom permutations (PRPs) like AES
- A lot of modes only use the forward direction, not making use of the invertibility
- In this case using a pseudorandom function (PRF) is often more secure

- Many symmetric cryptographic schemes are based on pseudorandom permutations (PRPs) like AES
- A lot of modes only use the forward direction, not making use of the invertibility
- In this case using a pseudorandom function (PRF) is often more secure
- Prominent example: CTR mode



$n/2$-bit security



n-bit security

PRP-to-PRF Methods

- We could design a dedicated PRF
- However, we have little understanding in how to design one

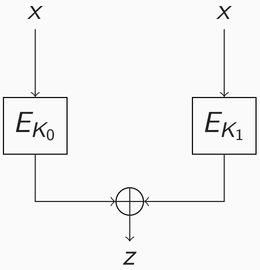
- We could design a dedicated PRF
- However, we have little understanding in how to design one
- Alternatively, we can design a PRP-to-PRF method

- We could design a dedicated PRF
- However, we have little understanding in how to design one
- Alternatively, we can design a PRP-to-PRF method
  - PRP-PRF switch: PRP behaves like a PRF up to the birthday bound

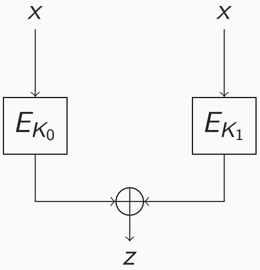
PRP-to-PRF Methods

- We could design a dedicated PRF
- However, we have little understanding in how to design one
- Alternatively, we can design a PRP-to-PRF method
  - PRP-PRF switch: PRP behaves like a PRF up to the birthday bound
  - Conversions like summation achieve beyond birthday bound security

Private Summation

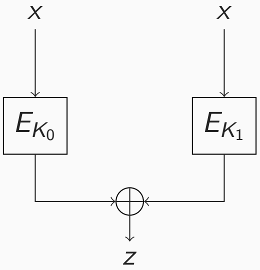


Private Summation



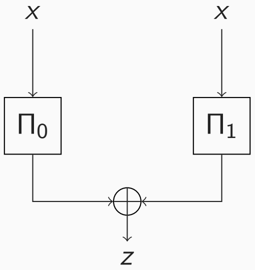
- Sums the output of two independent permutations

Private Summation

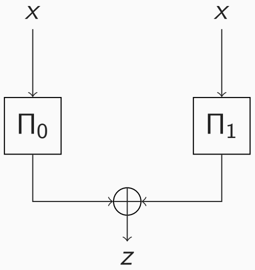


- Sums the output of two independent permutations
- Achieves n-bit security for private permutations

Public Summation

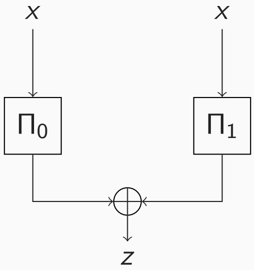


Public Summation



- In some situations the permutations are public

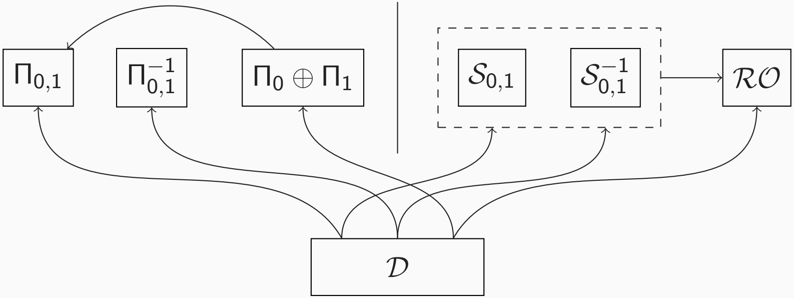
Public Summation



- In some situations the permutations are public
- Moves to indifferentiability setting

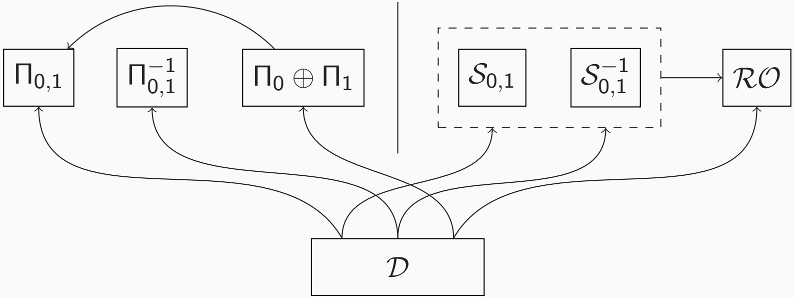
Indifferentiability

- Distinguisher $\mathcal{D}$ distinguishes between the real world and the ideal world



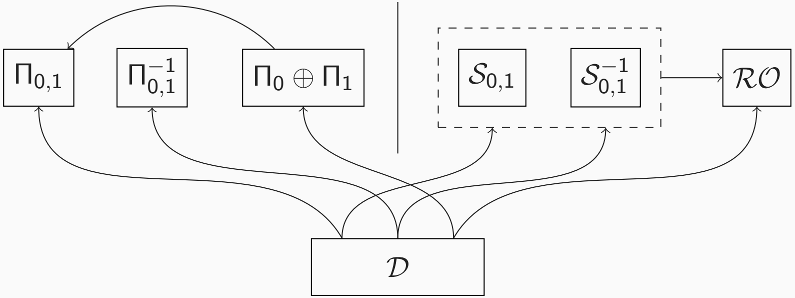
Indifferentiability

- Distinguisher $\mathcal{D}$ distinguishes between the real world and the ideal world
- Both primitive and construction queries



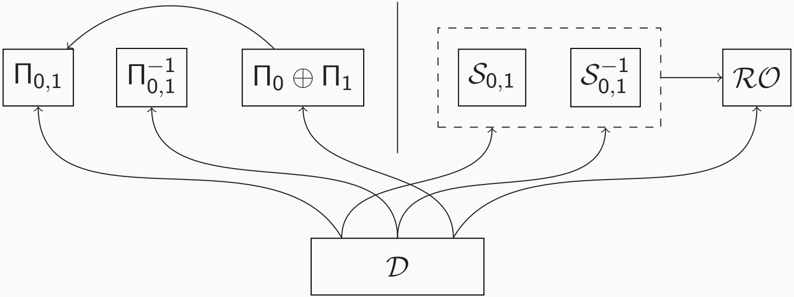
Indifferentiability

- Distinguisher $\mathcal{D}$ distinguishes between the real world and the ideal world
- Both primitive and construction queries
- Real world are public permutations $\Pi_{0,1}$ (primitive) and their summation $\Pi_0 \oplus \Pi_1$ (construction)



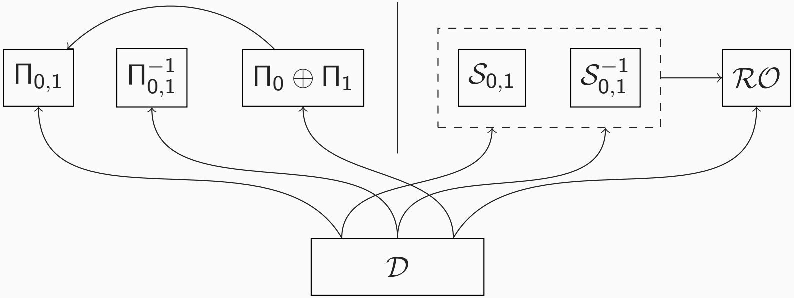
Indifferentiability

- Distinguisher $\mathcal{D}$ distinguishes between the real world and the ideal world
- Both primitive and construction queries
- Real world are public permutations $\Pi_{0,1}$ (primitive) and their summation $\Pi_0 \oplus \Pi_1$ (construction)
- Ideal world is a simulator $\mathcal{S}_{0,1}$ (primitive) and a random oracle $\mathcal{RO}$ (construction)



Indifferentiability

- Distinguisher $\mathcal{D}$ distinguishes between the real world and the ideal world
- Both primitive and construction queries
- Real world are public permutations $\Pi_{0,1}$ (primitive) and their summation $\Pi_0 \oplus \Pi_1$ (construction)
- Ideal world is a simulator $\mathcal{S}_{0,1}$ (primitive) and a random oracle $\mathcal{RO}$ (construction)
- Both forward and backward direction for the primitive queries



Previous works

- Three previous works about the indifferentiability of the sum of two permutations

- Three previous works about the indifferentiability of the sum of two permutations
  - Mandal et al. [MPN10] showed $2n/3$-bit security

Previous works

- Three previous works about the indifferentiability of the sum of two permutations
  - Mandal et al. [MPN10] showed $2n/3$-bit security
  - Mennink and Preneel [MP15] identified a flaw in [MPN10] and re-proved $(2n/3 - \log_2(n))$-bit security

Previous works

- Three previous works about the indifferentiability of the sum of two permutations
  - Mandal et al. [MPN10] showed $2n/3$-bit security
  - Mennink and Preneel [MP15] identified a flaw in [MPN10] and re-proved $(2n/3 - \log_2(n))$-bit security
  - Bhattacharya and Nandi [BN18] improved to n-bit security

Simulator (forward)

- All previous works use identical simulators up to negligible differences

Simulator (forward)

- All previous works use identical simulators up to negligible differences
- Simplified, the forward simulator $\mathcal{S}_0$ works as follows on input x:

Simulator (forward)

- All previous works use identical simulators up to negligible differences
- Simplified, the forward simulator $\mathcal{S}_0$ works as follows on input x:
  - Query the random oracle as $z = \mathcal{RO}(x)$

Simulator (forward)

- All previous works use identical simulators up to negligible differences
- Simplified, the forward simulator $\mathcal{S}_0$ works as follows on input x:
  - Query the random oracle as $z = \mathcal{RO}(x)$
  - Define the set of possible outputs as
    $Y = \{0,1\}^n \setminus (\text{range}(\mathcal{S}_0) \cup (\text{range}(\mathcal{S}_1) \oplus z))$

Simulator (forward)

- All previous works use identical simulators up to negligible differences
- Simplified, the forward simulator $\mathcal{S}_0$ works as follows on input x:
  - Query the random oracle as $z = \mathcal{RO}(x)$
  - Define the set of possible outputs as
    $Y = \{0,1\}^n \setminus (\text{range}(\mathcal{S}_0) \cup (\text{range}(\mathcal{S}_1) \oplus z))$
  - Sample a uniformly drawn output as $y_0 \xleftarrow{\$} Y$

Simulator (forward)

- All previous works use identical simulators up to negligible differences
- Simplified, the forward simulator $\mathcal{S}_0$ works as follows on input x:
  - Query the random oracle as $z = \mathcal{RO}(x)$
  - Define the set of possible outputs as
    $Y = \{0,1\}^n \setminus (\mathrm{range}(\mathcal{S}_0) \cup (\mathrm{range}(\mathcal{S}_1) \oplus z))$
  - Sample a uniformly drawn output as $y_0 \xleftarrow{\$} Y$
  - Return y_0

Simulator (inverse)

- Simplified, the inverse simulator $\mathcal{S}_0^{-1}$ works as follows on input y_0

Simulator (inverse)

- Simplified, the inverse simulator $\mathcal{S}_0^{-1}$ works as follows on input y_0
  - Sample a random fresh x

- Simplified, the inverse simulator $\mathcal{S}_0^{-1}$ works as follows on input y_0
  - Sample a random fresh x
  - Query the random oracle as $z = \mathcal{RO}(x)$

- Simplified, the inverse simulator $\mathcal{S}_0^{-1}$ works as follows on input y_0
  - Sample a random fresh x
  - Query the random oracle as $z = \mathcal{RO}(x)$
  - Check whether x is possible based on z:

- Simplified, the inverse simulator $\mathcal{S}_0^{-1}$ works as follows on input y_0
  - Sample a random fresh x
  - Query the random oracle as $z = \mathcal{RO}(x)$
  - Check whether x is possible based on z:
    - If it is possible, when $y_1 = y_0 \oplus z \notin \text{range}(\mathcal{S}_1)$, return x

Simulator (inverse)

- Simplified, the inverse simulator $\mathcal{S}_0^{-1}$ works as follows on input y_0
  - Sample a random fresh x
  - Query the random oracle as $z = \mathcal{RO}(x)$
  - Check whether x is possible based on z:
    - If it is possible, when $y_1 = y_0 \oplus z \notin \mathrm{range}(\mathcal{S}_1)$, return x
    - Otherwise, repeat the process up to ℓ times

Contributions

- Multiple contributions

Contributions

- Multiple contributions
  - All previous works are flawed

paper	security level	random range	sequentiality	fresh oracle
[MPN10]	$2n/3$	[MP15]	[Gun22]	—
[MP15]	$2n/3 - \log_2(n)$	—	[Gun22]	—
[BN18]	n	Ours	[Gun22]	Ours

Contributions

- Multiple contributions
  - All previous works are flawed

paper	security level	random range	sequentiality	fresh oracle
[MPN10]	$2n/3$	[MP15]	[Gun22]	—
[MP15]	$2n/3 - \log_2(n)$	—	[Gun22]	—
[BN18]	n	Ours	[Gun22]	Ours

  - Attack on standard simulator using $\mathcal{O}(2^{5n/6})$ queries

Contributions

- Multiple contributions
  - All previous works are flawed

paper	security level	random range	sequentiality	fresh oracle
[MPN10]	$2n/3$	[MP15]	[Gun22]	—
[MP15]	$2n/3 - \log_2(n)$	—	[Gun22]	—
[BN18]	n	Ours	[Gun22]	Ours

  - Attack on standard simulator using $\mathcal{O}(2^{5n/6})$ queries
  - Proof showing $(2n/3 - \log_2(n))$-bit security can be fixed using a new technique

Flaw 1: Random Range

- Let R_0 and R_1 be the ranges of the two primitives, i.e. in the real world we have

$$R_0 = \{ \Pi_0(x_i) \mid 1 \leqslant i \leqslant q \}$$
$$R_1 = \{ \Pi_1(x_i) \mid 1 \leqslant i \leqslant q \}$$

Flaw 1: Random Range

- Let R_0 and R_1 be the ranges of the two primitives, i.e. in the real world we have

$$R_0 = \{\, \Pi_0(x_i) \mid 1 \leqslant i \leqslant q \,\}$$
$$R_1 = \{\, \Pi_1(x_i) \mid 1 \leqslant i \leqslant q \,\}$$

- Then R_0 and R_1 are randomly distributed ✗

Flaw 1: Random Range

- Let R_0 and R_1 be the ranges of the two primitives, i.e. in the real world we have

$$R_0 = \{ \Pi_0(x_i) \mid 1 \leqslant i \leqslant q \}$$
$$R_1 = \{ \Pi_1(x_i) \mid 1 \leqslant i \leqslant q \}$$

- Then R_0 and R_1 are randomly distributed ✗
- Only true for forward queries, not backward ones

Flaw 1: Random Range

- Let R_0 and R_1 be the ranges of the two primitives, i.e. in the real world we have

$$R_0 = \{\, \Pi_0(x_i) \mid 1 \leqslant i \leqslant q \,\}$$
$$R_1 = \{\, \Pi_1(x_i) \mid 1 \leqslant i \leqslant q \,\}$$

- Then R_0 and R_1 are randomly distributed ✗
- Only true for forward queries, not backward ones
- Take the queries

$$\Pi_0^{-1}(0000), \Pi_0^{-1}(0001), \Pi_0^{-1}(0010), \Pi_0^{-1}(0011)$$

Flaw 1: Random Range

- Let R_0 and R_1 be the ranges of the two primitives, i.e. in the real world we have

$$R_0 = \{\, \Pi_0(x_i) \mid 1 \leqslant i \leqslant q \,\}$$
$$R_1 = \{\, \Pi_1(x_i) \mid 1 \leqslant i \leqslant q \,\}$$

- Then R_0 and R_1 are randomly distributed ✗
- Only true for forward queries, not backward ones
- Take the queries

$$\Pi_0^{-1}(0000), \Pi_0^{-1}(0001), \Pi_0^{-1}(0010), \Pi_0^{-1}(0011)$$

- Then $R_0 = \{0000, 0001, 0010, 0011\}$ is not random

Flaw 1: Random Range

- Let R_0 and R_1 be the ranges of the two primitives, i.e. in the real world we have

$$R_0 = \{\, \Pi_0(x_i) \mid 1 \leqslant i \leqslant q \,\}$$
$$R_1 = \{\, \Pi_1(x_i) \mid 1 \leqslant i \leqslant q \,\}$$

- Then R_0 and R_1 are randomly distributed ✗
- Only true for forward queries, not backward ones
- Take the queries

$$\Pi_0^{-1}(0000), \Pi_0^{-1}(0001), \Pi_0^{-1}(0010), \Pi_0^{-1}(0011)$$

- Then $R_0 = \{0000, 0001, 0010, 0011\}$ is not random
- Fundamental problem, invalidating [MPN10, BN18]

Flaw 2: Sequentiality

- Modify the distinguisher $\mathcal{D}$ to an equivalent one $\mathcal{D}'$: ✓

Primitive	Construction

- Modify the distinguisher $\mathcal{D}$ to an equivalent one $\mathcal{D}'$: ✓
  - Interact like $\mathcal{D}$
    ($x_{\text{min}} = x_1$ if $z_1 < z_2$ else x_2)

Primitive	Construction
	$\mathcal{RO}(x_1) = z_1$
	$\mathcal{RO}(x_2) = z_2$
$\mathcal{S}_0(x_{\text{min}}) = y_{\text{min}}$	
$\mathcal{S}_0(x_{\text{max}}) = y_{\text{max}}$	

Flaw 2: Sequentiality

- Modify the distinguisher $\mathcal{D}$ to an equivalent one $\mathcal{D}'$: ✓
  - Interact like $\mathcal{D}$ ($x_{\min} = x_1$ if $z_1 < z_2$ else x_2)
  - Add verification queries for all construction queries

Primitive	Construction
	$\mathcal{RO}(x_1) = z_1$
	$\mathcal{RO}(x_2) = z_2$
$\mathcal{S}_0(x_{\min}) = y_{\min}$	
$\mathcal{S}_0(x_{\max}) = y_{\max}$	
$\mathcal{S}_1(x_1) = y_1 \oplus z_1$	
$\mathcal{S}_1(x_2) = y_2 \oplus z_2$	

Flaw 2: Sequentiality

- Modify the distinguisher $\mathcal{D}$ to an equivalent one $\mathcal{D}'$: ✓
  - Interact like $\mathcal{D}$ ($x_{\min} = x_1$ if $z_1 < z_2$ else x_2)
  - Add verification queries for all construction queries
  - Output the same decision as $\mathcal{D}$

Primitive	Construction
	$\mathcal{RO}(x_1) = z_1$
	$\mathcal{RO}(x_2) = z_2$
$\mathcal{S}_0(x_{\min}) = y_{\min}$	
$\mathcal{S}_0(x_{\max}) = y_{\max}$	
$\mathcal{S}_1(x_1) = y_1 \oplus z_1$	
$\mathcal{S}_1(x_2) = y_2 \oplus z_2$	

$$y_{\min} \overset{?}{<} y_{\max}$$

Flaw 2: Sequentiality

- Modify the distinguisher $\mathcal{D}$ to an equivalent one $\mathcal{D}'$: ✓
  - Interact like $\mathcal{D}$ ($x_{\min} = x_1$ if $z_1 < z_2$ else x_2)
  - Add verification queries for all construction queries
  - Output the same decision as $\mathcal{D}$
- Note that these queries contain duplicate information ✓

Primitive	Construction
	$\mathcal{RO}(x_1) = z_1$
	$\mathcal{RO}(x_2) = z_2$
$\mathcal{S}_0(x_{\min}) = y_{\min}$	
$\mathcal{S}_0(x_{\max}) = y_{\max}$	
$\mathcal{S}_1(x_1) = y_1 \oplus z_1$	
$\mathcal{S}_1(x_2) = y_2 \oplus z_2$	

$$y_{\min} \overset{?}{<} y_{\max}$$

Flaw 2: Sequentiality

- Modify the distinguisher $\mathcal{D}$ to an equivalent one $\mathcal{D}'$: ✓
  - Interact like $\mathcal{D}$ ($x_{\min} = x_1$ if $z_1 < z_2$ else x_2)
  - Add verification queries for all construction queries
  - Output the same decision as $\mathcal{D}$
- Note that these queries contain duplicate information ✓
- Ignore the construction queries, leaving only the primitive ones ✗

Primitive	Construction
$\mathcal{S}_0(x_{\min}) = y_{\min}$	
$\mathcal{S}_0(x_{\max}) = y_{\max}$	
$\mathcal{S}_1(x_1) = y_1 \oplus z_1$	
$\mathcal{S}_1(x_2) = y_2 \oplus z_2$	

$$y_{\min} \overset{?}{<} y_{\max}$$

Flaw 2: Sequentiality

- Modify the distinguisher $\mathcal{D}$ to an equivalent one $\mathcal{D}'$: ✓
  - Interact like $\mathcal{D}$ ($x_{\min} = x_1$ if $z_1 < z_2$ else x_2)
  - Add verification queries for all construction queries
  - Output the same decision as $\mathcal{D}$
- Note that these queries contain duplicate information ✓
- Ignore the construction queries, leaving only the primitive ones ✗
- Disregards that the construction queries can have influence on later queries

Primitive	Construction
$\mathcal{S}_0(x_{\min}) = y_{\min}$	
$\mathcal{S}_0(x_{\max}) = y_{\max}$	
$\mathcal{S}_1(x_1) = y_1 \oplus z_1$	
$\mathcal{S}_1(x_2) = y_2 \oplus z_2$	

$$y_{\min} \overset{?}{<} y_{\max}$$

Flaw 2: Sequentiality ctd.

- There is an alternative modification with the same flaw

Primitive	Construction
	$\mathcal{RO}(x_1) = z_1$
	$\mathcal{RO}(x_2) = z_2$
$\mathcal{S}_0(x_{min}) = y_{min}$	
$\mathcal{S}_0(x_{max}) = y_{max}$	

$$y_{min} \overset{?}{<} y_{max}$$

Flaw 2: Sequentiality ctd.

- There is an alternative modification with the same flaw
- Execute the verification queries at the same time as the construction queries ✗

Primitive	Construction
$\mathcal{S}_0(x_1) = y_1$	
$\mathcal{S}_1(x_1) = y_1 \oplus z_1$	
$\mathcal{S}_0(x_2) = y_2$	
$\mathcal{S}_1(x_2) = y_2 \oplus z_2$	
$\mathcal{S}_0(x_{\min}) = y_{\min}$	
$\mathcal{S}_0(x_{\max}) = y_{\max}$	

$$y_{\min} \overset{?}{<} y_{\max}$$

Flaw 2: Sequentiality ctd.

- There is an alternative modification with the same flaw
- Execute the verification queries at the same time as the construction queries ✗
- This changes the order of the primitive queries, which does influence its behavior

Primitive	Construction
$\mathcal{S}_0(x_1) = y_1$	
$\mathcal{S}_1(x_1) = y_1 \oplus z_1$	
$\mathcal{S}_0(x_2) = y_2$	
$\mathcal{S}_1(x_2) = y_2 \oplus z_2$	
$\mathcal{S}_0(x_{\min}) = y_{\min}$	
$\mathcal{S}_0(x_{\max}) = y_{\max}$	

$$y_{\min} \overset{?}{<} y_{\max}$$

Flaw 2: Sequentiality ctd.

- There is an alternative modification with the same flaw
- Execute the verification queries at the same time as the construction queries ✗
- This changes the order of the primitive queries, which does influence its behavior
- Works in the weaker sequential indifferentiability setting, where all primitive queries have to be made before the construction queries

Primitive	Construction
$\mathcal{S}_0(x_1) = y_1$	
$\mathcal{S}_1(x_1) = y_1 \oplus z_1$	
$\mathcal{S}_0(x_2) = y_2$	
$\mathcal{S}_1(x_2) = y_2 \oplus z_2$	
$\mathcal{S}_0(x_{min}) = y_{min}$	
$\mathcal{S}_0(x_{max}) = y_{max}$	

$$y_{min} \overset{?}{<} y_{max}$$

Stateless versus Stateful

- All previous works do one of these transformations

Stateless versus Stateful

- All previous works do one of these transformations
- Simulator viewed as a stateless primitive

Stateless versus Stateful

- All previous works do one of these transformations
- Simulator viewed as a stateless primitive
- A stateless primitive can be implemented by drawing all randomness at the start

Stateless versus Stateful

- All previous works do one of these transformations
- Simulator viewed as a stateless primitive
- A stateless primitive can be implemented by drawing all randomness at the start
- Most primitives are stateless: random permutations, random function, random oracle, etc.

Stateless versus Stateful

- All previous works do one of these transformations
- Simulator viewed as a stateless primitive
- A stateless primitive can be implemented by drawing all randomness at the start
- Most primitives are stateless: random permutations, random function, random oracle, etc.
- The simulator is stateful, making analysis more difficult

Reordering Queries

- A stateless primitive allows queries to be made in any order: $P(x_1), P(x_2)$ has the same distribution as $P(x_2), P(x_1)$, simplifying analysis

Reordering Queries

- A stateless primitive allows queries to be made in any order: $P(x_1), P(x_2)$ has the same distribution as $P(x_2), P(x_1)$, simplifying analysis
- This same property is assumed for the simulator and is the core of the flaw

Reordering Queries

- A stateless primitive allows queries to be made in any order: $P(x_1), P(x_2)$ has the same distribution as $P(x_2), P(x_1)$, simplifying analysis
- This same property is assumed for the simulator and is the core of the flaw
- The simulator is stateful and does not have this same behavior

Reordering Queries

- A stateless primitive allows queries to be made in any order: $P(x_1), P(x_2)$ has the same distribution as $P(x_2), P(x_1)$, simplifying analysis
- This same property is assumed for the simulator and is the core of the flaw
- The simulator is stateful and does not have this same behavior
- We show that the simulator partly has this property

Reordering Queries

- A stateless primitive allows queries to be made in any order: $P(x_1), P(x_2)$ has the same distribution as $P(x_2), P(x_1)$, simplifying analysis
- This same property is assumed for the simulator and is the core of the flaw
- The simulator is stateful and does not have this same behavior
- We show that the simulator partly has this property
- Queries can be reordered as necessary up to $2n/3$-bit security

Reordering Queries

- A stateless primitive allows queries to be made in any order: $P(x_1), P(x_2)$ has the same distribution as $P(x_2), P(x_1)$, simplifying analysis
- This same property is assumed for the simulator and is the core of the flaw
- The simulator is stateful and does not have this same behavior
- We show that the simulator partly has this property
- Queries can be reordered as necessary up to $2n/3$-bit security
- Re-establishes regular indifferentiability with $(2n/3 - \log_2(n))$-bit security using [MP15] for sequential indifferentiability

Flaw 3: Fresh Oracle

- A value returned from the random oracle is uniformly at random distributed ✗

Flaw 3: Fresh Oracle

- A value returned from the random oracle is uniformly at random distributed ✗
- Does not hold due to the behavior of the inverse simulator

Flaw 3: Fresh Oracle

- A value returned from the random oracle is uniformly at random distributed ✗
- Does not hold due to the behavior of the inverse simulator
- Comparison to illustrate the problem

Comparison: Bag of M&M's

- Consider a bag of 10 colored M&M's

Comparison: Bag of M&M's

- Consider a bag of 10 colored M&M's
- They are uniformly sampled from 5 colors: **red**, **brown**, **yellow**, **green** and **blue**

Comparison: Bag of M&M's

- Consider a bag of 10 colored M&M's
- They are uniformly sampled from 5 colors: **red**, **brown**, **yellow**, **green** and **blue**
- A randomly drawn M&M has a probability of $1/5$ of being a specific color, even after other draws

Comparison: Bag of M&M's

- Consider a bag of 10 colored M&M's
- They are uniformly sampled from 5 colors: **red**, **brown**, **yellow**, **green** and **blue**
- A randomly drawn M&M has a probability of $1/5$ of being a specific color, even after other draws
- Suppose you do not like brown M&M's and do the following when grabbing one:

Comparison: Bag of M&M's

- Consider a bag of 10 colored M&M's
- They are uniformly sampled from 5 colors: **red**, **brown**, **yellow**, **green** and **blue**
- A randomly drawn M&M has a probability of $1/5$ of being a specific color, even after other draws
- Suppose you do not like brown M&M's and do the following when grabbing one:
  - If it is **brown**: redraw (can be brown), put the original M&M back

Comparison: Bag of M&M's

- Consider a bag of 10 colored M&M's
- They are uniformly sampled from 5 colors: **red**, **brown**, **yellow**, **green** and **blue**
- A randomly drawn M&M has a probability of $1/5$ of being a specific color, even after other draws

- Suppose you do not like brown M&M's and do the following when grabbing one:
  - If it is **brown**: redraw (can be brown), put the original M&M back
  - If it is **any other colored M&M**: eat it

Comparison: Bag of M&M's

- Consider a bag of 10 colored M&M's
- They are uniformly sampled from 5 colors: **red**, **brown**, **yellow**, **green** and **blue**
- A randomly drawn M&M has a probability of $1/5$ of being a specific color, even after other draws
- Suppose you do not like brown M&M's and do the following when grabbing one:
  - If it is **brown**: redraw (can be brown), put the original M&M back
  - If it is **any other colored M&M**: eat it
- After this process, the probability that an M&M in the bag is brown becomes:

$$\frac{4}{5} \cdot \frac{1}{5} + \frac{1}{5} \cdot \left(\frac{8}{9} \cdot \frac{1}{5} + \frac{1}{9} \cdot 1 \right) = \frac{49}{225} > \frac{45}{225} = \frac{1}{5}$$

Comparison: Bag of M&M's (ctd.)

- Similar issue is present in [BN18]

- Similar issue is present in [BN18]
- Other works [MPN10, MP15] acknowledge the difference

Comparison: Bag of M&M's (ctd.)

- Similar issue is present in [BN18]
- Other works [MPN10, MP15] acknowledge the difference
- Partly responsible for limited $2n/3$-bit security in those works

Comparison: Bag of M&M's (ctd.)

- Similar issue is present in [BN18]
- Other works [MPN10, MP15] acknowledge the difference
- Partly responsible for limited $2n/3$-bit security in those works
- We give an attack that shows that this difference matters for more than $3n/4$-bit security

Attack: Standard Simulator Limited to $5n/6$-bit Security

- Recall that the forward simulator selects its output y_0 uniformly from all possibilities $Y = \{0,1\}^n \setminus (\mathrm{range}(\mathcal{S}_0) \cup (\mathrm{range}(\mathcal{S}_1) \oplus z))$

Attack: Standard Simulator Limited to $5n/6$-bit Security

- Recall that the forward simulator selects its output y_0 uniformly from all possibilities $Y = \{0,1\}^n \setminus (\mathrm{range}(\mathcal{S}_0) \cup (\mathrm{range}(\mathcal{S}_1) \oplus z))$
- Surprisingly, in some cases the sampling in the real world does not behave uniformly

Attack: Standard Simulator Limited to $5n/6$-bit Security

- Recall that the forward simulator selects its output y_0 uniformly from all possibilities $Y = \{0,1\}^n \setminus (\mathrm{range}(\mathcal{S}_0) \cup (\mathrm{range}(\mathcal{S}_1) \oplus z))$
- Surprisingly, in some cases the sampling in the real world does not behave uniformly
- Gives rise to an attack using $\mathcal{O}(2^{5n/6})$ queries

Attack: Standard Simulator Limited to $5n/6$-bit Security

- Recall that the forward simulator selects its output y_0 uniformly from all possibilities $Y = \{0,1\}^n \setminus (\mathrm{range}(\mathcal{S}_0) \cup (\mathrm{range}(\mathcal{S}_1) \oplus z))$
- Surprisingly, in some cases the sampling in the real world does not behave uniformly
- Gives rise to an attack using $\mathcal{O}(2^{5n/6})$ queries
- Maybe possible to fix with a biased simulator, but gets very complicated

Conclusion

- An established beyond birthday bound PRP-to-PRF conversion is the sum of permutations

Conclusion

- An established beyond birthday bound PRP-to-PRF conversion is the sum of permutations
- All previous works on its indifferentiability are flawed

Conclusion

- An established beyond birthday bound PRP-to-PRF conversion is the sum of permutations
- All previous works on its indifferentiability are flawed
- We show limitations for many different approaches

Conclusion

- An established beyond birthday bound PRP-to-PRF conversion is the sum of permutations
- All previous works on its indifferentiability are flawed
- We show limitations for many different approaches
- Also positive result: regular indifferentiability with $(2n/3 - \log_2(n))$-bit security

Conclusion

- An established beyond birthday bound PRP-to-PRF conversion is the sum of permutations
- All previous works on its indifferentiability are flawed
- We show limitations for many different approaches
- Also positive result: regular indifferentiability with $(2n/3 - \log_2(n))$-bit security

Thank you for your attention!

Srimanta Bhattacharya and Mridul Nandi.
Full Indifferentiable Security of the Xor of Two or More Random Permutations Using the χ^2 Method.
In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 387–412. Springer, 2018.

Aldo Gunsing.
Block-cipher-based tree hashing.
Springer-Verlag, 2022.

Bart Mennink and Bart Preneel.
On the XOR of Multiple Random Permutations.

In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *Applied Cryptography and Network Security - 13th International Conference, ACNS 2015, New York, NY, USA, June 2-5, 2015, Revised Selected Papers*, volume 9092 of *Lecture Notes in Computer Science*, pages 619–634. Springer, 2015.

Avradip Mandal, Jacques Patarin, and Valérie Nachef.
Indifferentiability beyond the Birthday Bound for the Xor of Two Public Random Permutations.
In Guang Gong and Kishan Chand Gupta, editors, *Progress in Cryptology - INDOCRYPT 2010 - 11th International Conference on Cryptology in India, Hyderabad, India, December 12-15, 2010. Proceedings*, volume 6498 of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2010.