Radbound University

Differential Trail Search in Cryptographic Primitives with Big-Circle Chi

Application to SUBTERRANEAN

Alireza Mehrdad, Silvia Mella, Lorenzo Grassi and Joan Daemen

March 22, 2023

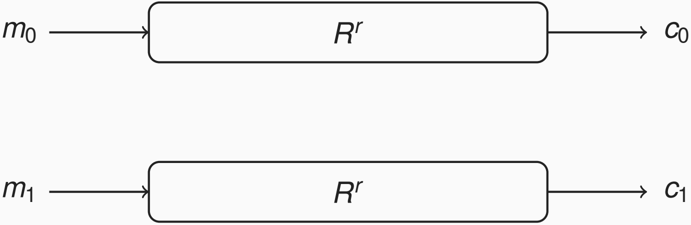
ESCADA

Introduction

Differential cryptanalysis

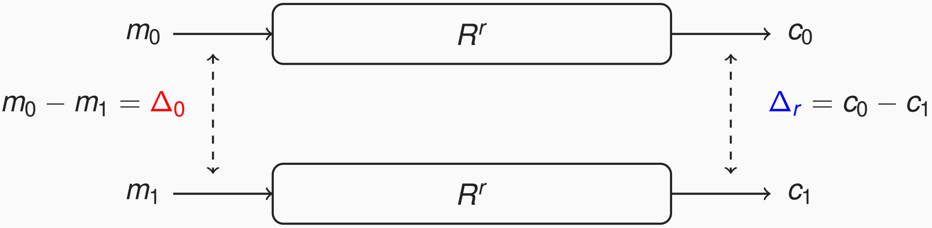
$$m_0 \longrightarrow \boxed{\quad R^r \quad} \longrightarrow c_0$$

Differential cryptanalysis

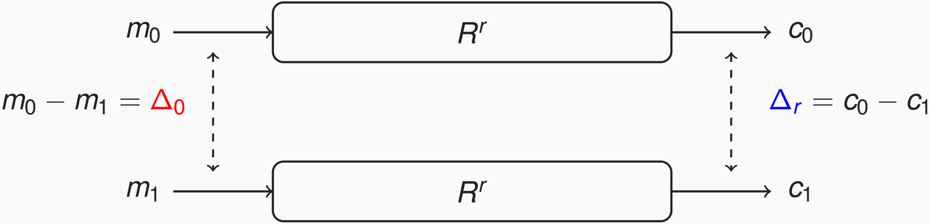


Differential cryptanalysis

Differential cryptanalysis

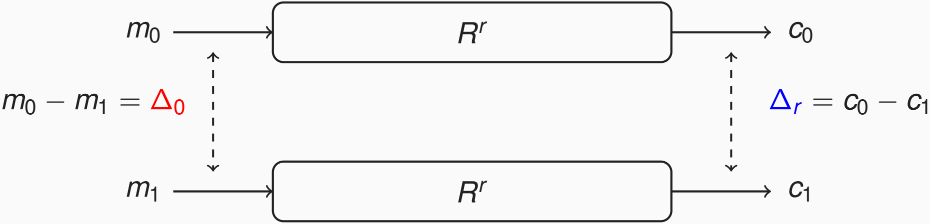


Differential cryptanalysis



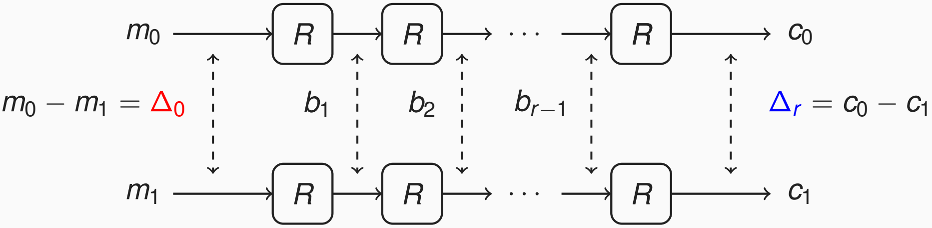
- Security: $\max \mathrm{DP}(\Delta_0 \to \Delta_r)$

Differential cryptanalysis

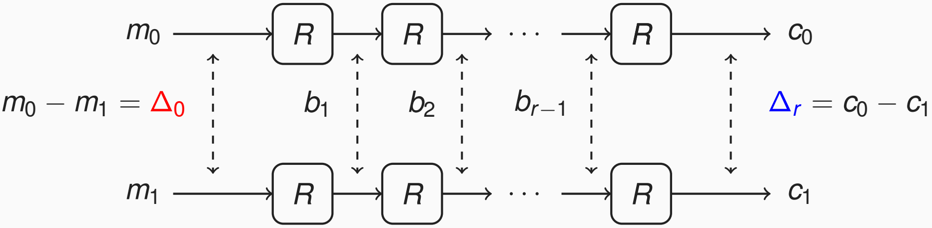


- Security: $\max \mathrm{DP}(\Delta_0 \rightarrow \Delta_r) \rightarrow$ It is hard to determine

Differential cryptanalysis

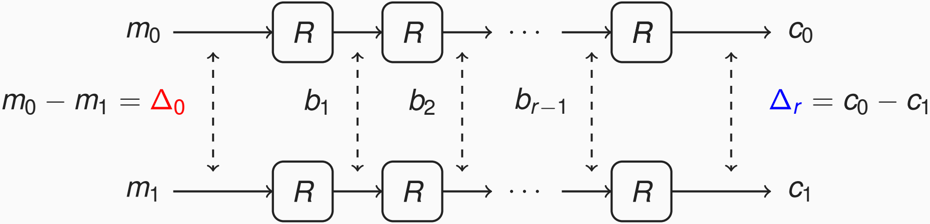


- Security: $\max \mathrm{DP}(\Delta_0 \to \Delta_r) \to$ It is hard to determine



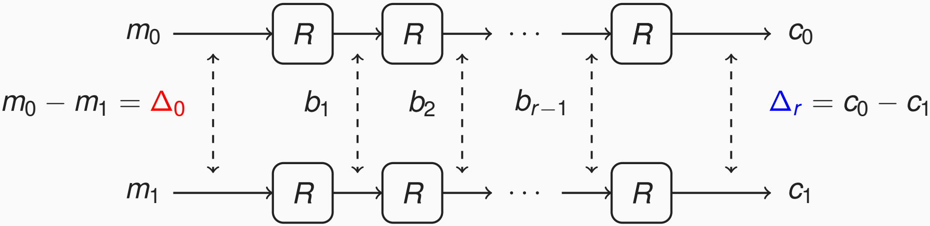
- Security: $\max \mathrm{DP}(\Delta_0 \to \Delta_r) \to$ It is hard to determine
- $Q_r : \Delta_0 \to b_1 \to b_2 \to \cdots \to b_{r-1} \to \Delta_r$ is a differential trail

Differential cryptanalysis

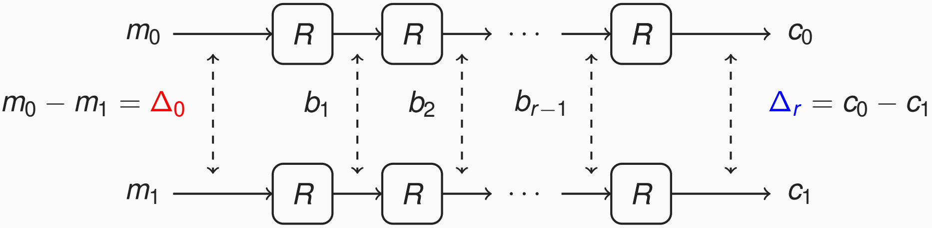


- Security: $\max \mathrm{DP}(\Delta_0 \to \Delta_r) \to$ It is hard to determine
- $Q_r : \Delta_0 \to b_1 \to b_2 \to \cdots \to b_{r-1} \to \Delta_r$ is a differential trail
  - $\mathrm{DP}(Q_r) \approx \mathrm{DP}(\Delta_0 \to b_1) \times \mathrm{DP}(b_1 \to b_2) \times \ldots \times \mathrm{DP}(b_{r-1} \to \Delta_r)$

Differential cryptanalysis

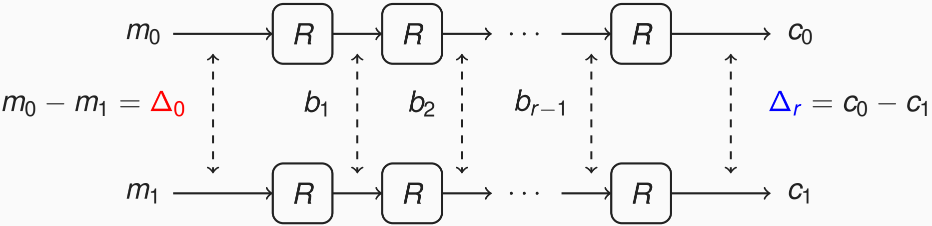


- Security: $\max \mathrm{DP}(\Delta_0 \to \Delta_r) \to$ It is hard to determine
- $Q_r : \Delta_0 \to b_1 \to b_2 \to \cdots \to b_{r-1} \to \Delta_r$ is a differential trail
  - $\mathrm{DP}(Q_r) \approx \mathrm{DP}(\Delta_0 \to b_1) \times \mathrm{DP}(b_1 \to b_2) \times \ldots \times \mathrm{DP}(b_{r-1} \to \Delta_r)$
  - $\mathrm{DP}(\Delta_0 \to \Delta_r) \approx \mathrm{DP}(Q_r)$



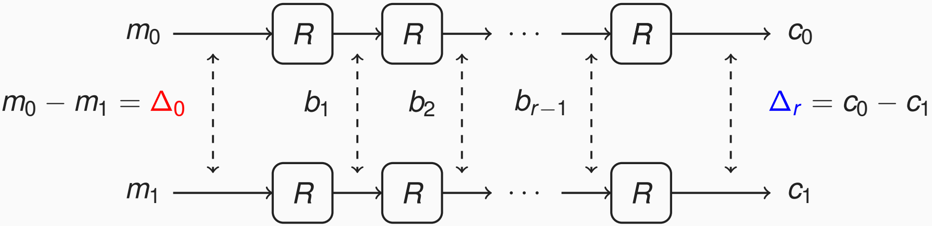
- Security: $\max \mathrm{DP}(\Delta_0 \to \Delta_r) \to$ It is hard to determine
- $Q_r : \Delta_0 \to b_1 \to b_2 \to \cdots \to b_{r-1} \to \Delta_r$ is a differential trail
  - $\mathrm{DP}(Q_r) \approx \mathrm{DP}(\Delta_0 \to b_1) \times \mathrm{DP}(b_1 \to b_2) \times \ldots \times \mathrm{DP}(b_{r-1} \to \Delta_r)$
  - $\mathrm{DP}(\Delta_0 \to \Delta_r) \approx \mathrm{DP}(Q_r)$
  - $w(b_i \to b_{i+1}) = -\log_2 \mathrm{DP}(b_i \to b_{i+1})$

Differential cryptanalysis



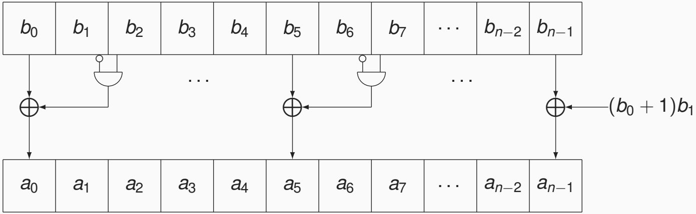
- Security: $\max \mathrm{DP}(\Delta_0 \to \Delta_r) \to$ It is hard to determine
- $Q_r : \Delta_0 \to b_1 \to b_2 \to \cdots \to b_{r-1} \to \Delta_r$ is a differential trail
  - $\mathrm{DP}(Q_r) \approx \mathrm{DP}(\Delta_0 \to b_1) \times \mathrm{DP}(b_1 \to b_2) \times \ldots \times \mathrm{DP}(b_{r-1} \to \Delta_r)$
  - $\mathrm{DP}(\Delta_0 \to \Delta_r) \approx \mathrm{DP}(Q_r)$
  - $w(b_i \to b_{i+1}) = -\log_2 \mathrm{DP}(b_i \to b_{i+1}) \to$ (e. g. $\mathrm{DP} = 2^{-n} \to w = n$)

Differential cryptanalysis



- Security: $\max \mathrm{DP}(\Delta_0 \to \Delta_r) \to$ It is hard to determine
- $Q_r : \Delta_0 \to b_1 \to b_2 \to \cdots \to b_{r-1} \to \Delta_r$ is a differential trail
  - $\mathrm{DP}(Q_r) \approx \mathrm{DP}(\Delta_0 \to b_1) \times \mathrm{DP}(b_1 \to b_2) \times \ldots \times \mathrm{DP}(b_{r-1} \to \Delta_r)$
  - $\mathrm{DP}(\Delta_0 \to \Delta_r) \approx \mathrm{DP}(Q_r)$
  - $w(b_i \to b_{i+1}) = -\log_2 \mathrm{DP}(b_i \to b_{i+1}) \to$ (e. g. $\mathrm{DP} = 2^{-n} \to w = n$)
  - $w(Q_r) = w(\Delta_0 \to b_1) + w(b_1 \to b_2) + \ldots + w(b_{r-1} \to \Delta_r)$

χ_n transformation

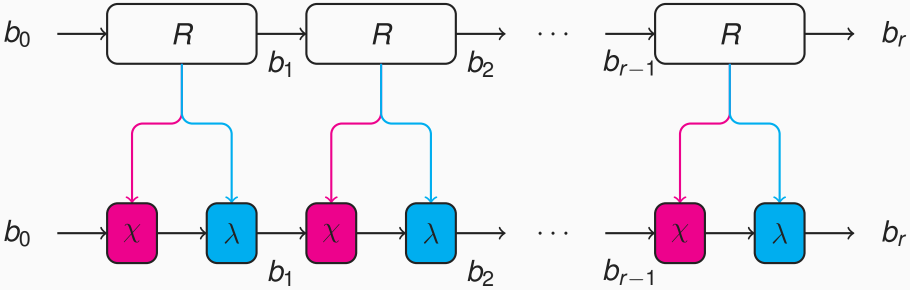


Generating r-round Differential trails

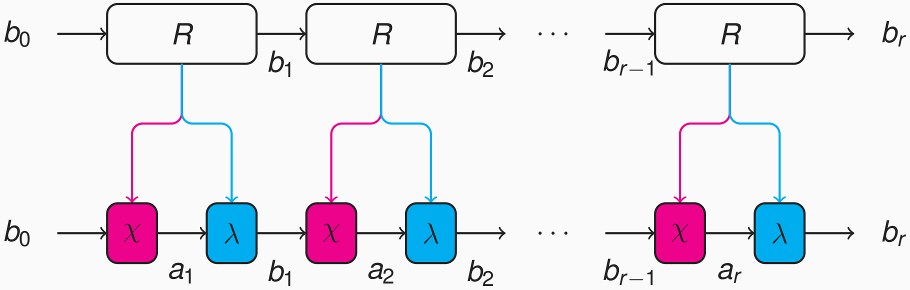
r-round differential trail

$$b_0 \xrightarrow{R} b_1 \xrightarrow{R} b_2 \longrightarrow \cdots \longrightarrow b_{r-1} \xrightarrow{R} b_r$$

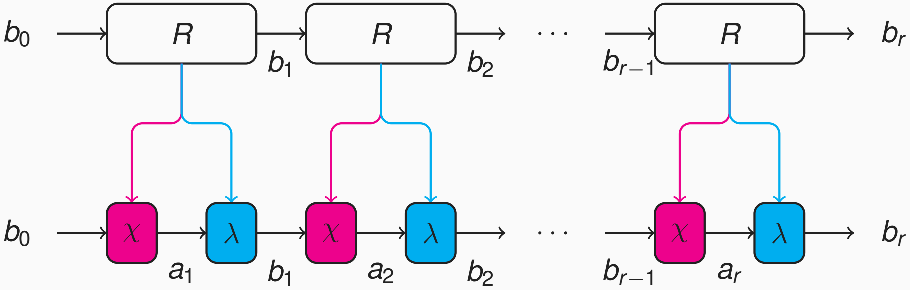
r-round differential trail



r-round differential trail

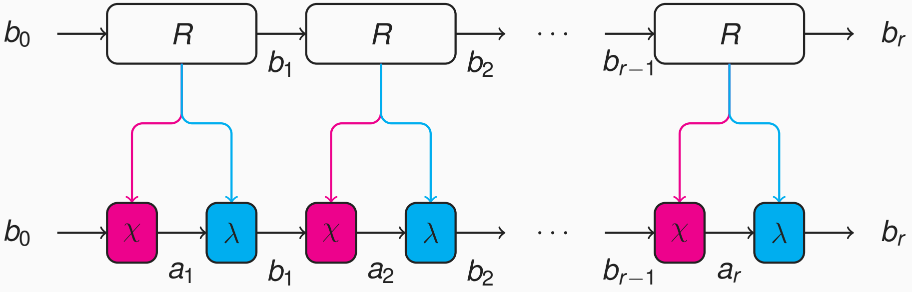


r-round differential trail



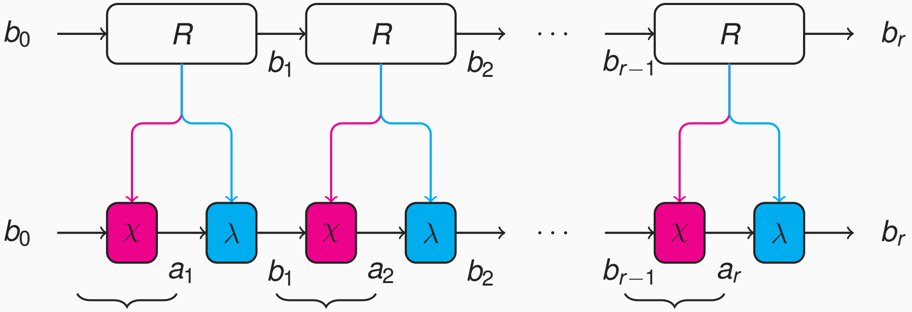
- $\mathrm{DP}(a_i \xrightarrow{\lambda} b_i) = 1 = 2^0$

r-round differential trail



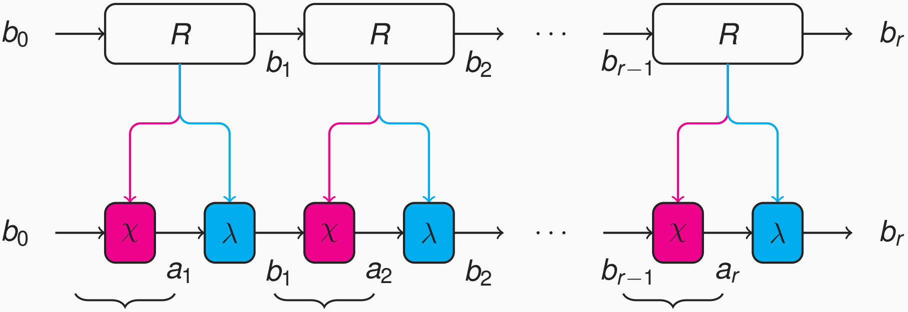
- $\mathrm{DP}(a_i \xrightarrow{\lambda} b_i) = 1 = 2^0 \Rightarrow w(a_i \xrightarrow{\lambda} b_i) = 0$

r-round differential trail



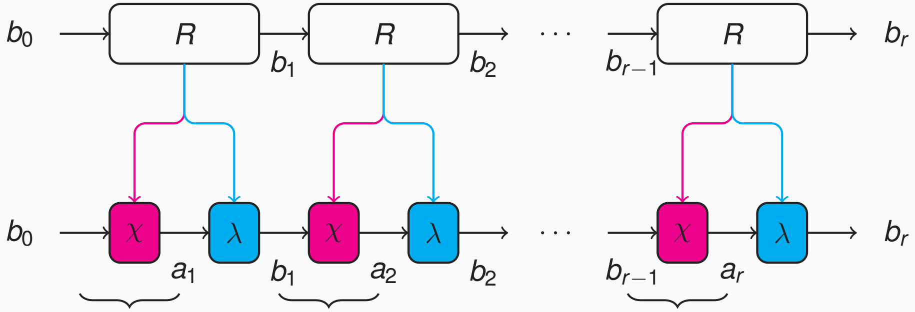
- $\mathrm{DP}(a_i \xrightarrow{\lambda} b_i) = 1 = 2^0 \Rightarrow w(a_i \xrightarrow{\lambda} b_i) = 0$
- $w(Q_r) = \sum_{i=0}^{r-1} w(b_i \xrightarrow{\chi} a_{i+1})$

r-round differential trail



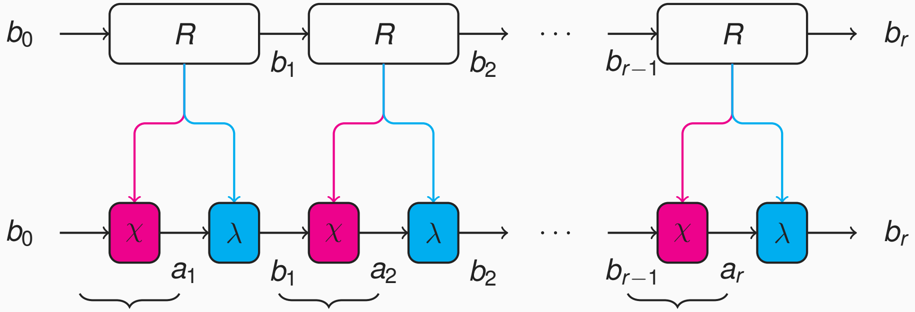
- $\mathrm{DP}(a_i \xrightarrow{\lambda} b_i) = 1 = 2^0 \Rightarrow w(a_i \xrightarrow{\lambda} b_i) = 0$
- $w(Q_r) = \sum_{i=0}^{r-1} w(b_i \xrightarrow{\chi} a_{i+1})$
- Goal: r-round trails with the minimum weight (maximum DP)

r-round differential trail



- $\mathrm{DP}(a_i \xrightarrow{\lambda} b_i) = 1 = 2^0 \Rightarrow w(a_i \xrightarrow{\lambda} b_i) = 0$
- $w(Q_r) = \sum_{i=0}^{r-1} w(b_i \xrightarrow{\chi} a_{i+1})$
- Goal: r-round trails with the minimum weight (maximum DP)
- Not efficient to start with r-round trail

r-round differential trail



- $\mathrm{DP}(a_i \xrightarrow{\lambda} b_i) = 1 = 2^0 \Rightarrow w(a_i \xrightarrow{\lambda} b_i) = 0$
- $w(Q_r) = \sum_{i=0}^{r-1} w(b_i \xrightarrow{\chi} a_{i+1})$
- Goal: r-round trails with the minimum weight (maximum DP)
- Not efficient to start with r-round trail $\rightarrow$ 2-round trails up to weight T_2

2-round trails and trail cores

- 2-round trail: $b_i \xrightarrow{\chi} a_{i+1} \xrightarrow{\lambda} b_{i+1} \xrightarrow{\chi} a_{i+2} \xrightarrow{\lambda} b_{i+2}$

2-round trails and trail cores

- 2-round trail: $b_i \xrightarrow{\chi} a_{i+1} \xrightarrow{\lambda} b_{i+1} \xrightarrow{\chi} a_{i+2} \xrightarrow{\lambda} b_{i+2}$
- Since χ has degree 2, the weight is determined by just its input

2-round trails and trail cores

- 2-round trail: $b_i \xrightarrow{\chi} a_{i+1} \xrightarrow{\lambda} b_{i+1} \xrightarrow{\chi} a_{i+2} \xrightarrow{\lambda} b_{i+2}$
- Since χ has degree 2, the weight is determined by just its input
  - $w(Q_2) = w(b_i \xrightarrow{\chi} a_{i+1}) + w(b_{i+1} \xrightarrow{\chi} a_{i+2})$

2-round trails and trail cores

- 2-round trail: $b_i \xrightarrow{\chi} a_{i+1} \xrightarrow{\lambda} b_{i+1} \xrightarrow{\chi} a_{i+2} \xrightarrow{\lambda} b_{i+2}$
- Since χ has degree 2, the weight is determined by just its input
    - $w(Q_2) = w(b_i \xrightarrow{\chi} a_{i+1}) + w(b_{i+1} \xrightarrow{\chi} a_{i+2}) = w(b_i) + w(b_{i+1})$

2-round trails and trail cores

- 2-round trail: $b_i \xrightarrow{\chi} a_{i+1} \xrightarrow{\lambda} b_{i+1} \xrightarrow{\chi} a_{i+2} \xrightarrow{\lambda} b_{i+2}$
- Since χ has degree 2, the weight is determined by just its input
  - $w(Q_2) = w(b_i \xrightarrow{\chi} a_{i+1}) + w(b_{i+1} \xrightarrow{\chi} a_{i+2}) = w(b_i) + w(b_{i+1})$
- We can ignore a_{i+2} and b_{i+2} and only build the following
  - $b_i \xrightarrow{\chi} a_{i+1} \xrightarrow{\lambda} b_{i+1} \xrightarrow{\chi}$

2-round trails and trail cores

- 2-round trail: $b_i \xrightarrow{\chi} a_{i+1} \xrightarrow{\lambda} b_{i+1} \xrightarrow{\chi} a_{i+2} \xrightarrow{\lambda} b_{i+2}$
- Since χ has degree 2, the weight is determined by just its input
  - $w(Q_2) = w(b_i \xrightarrow{\chi} a_{i+1}) + w(b_{i+1} \xrightarrow{\chi} a_{i+2}) = w(b_i) + w(b_{i+1})$
- We can ignore a_{i+2} and b_{i+2} and only build the following
  - $b_i \xrightarrow{\chi} a_{i+1} \xrightarrow{\lambda} b_{i+1} \xrightarrow{\chi}$
- Minimum reverse weight $w_{rev}(a_{i+1})$: minimum over the weight of all compatible b_i's
- We introduced a new method to compute the $w_{rev}(a_{i+1})$

2-round trails and trail cores

- 2-round trail: $b_i \xrightarrow{\chi} a_{i+1} \xrightarrow{\lambda} b_{i+1} \xrightarrow{\chi} a_{i+2} \xrightarrow{\lambda} b_{i+2}$
- Since χ has degree 2, the weight is determined by just its input
  - $w(Q_2) = w(b_i \xrightarrow{\chi} a_{i+1}) + w(b_{i+1} \xrightarrow{\chi} a_{i+2}) = w(b_i) + w(b_{i+1})$
- We can ignore a_{i+2} and b_{i+2} and only build the following
  - $b_i \xrightarrow{\chi} a_{i+1} \xrightarrow{\lambda} b_{i+1} \xrightarrow{\chi}$
- Minimum reverse weight $w_{rev}(a_{i+1})$: minimum over the weight of all compatible b_i's
- We introduced a new method to compute the $w_{rev}(a_{i+1}) \rightarrow$ ignore b_i

2-round trails and trail cores

- 2-round trail: $b_i \xrightarrow{\chi} a_{i+1} \xrightarrow{\lambda} b_{i+1} \xrightarrow{\chi} a_{i+2} \xrightarrow{\lambda} b_{i+2}$
- Since χ has degree 2, the weight is determined by just its input
  - $w(Q_2) = w(b_i \xrightarrow{\chi} a_{i+1}) + w(b_{i+1} \xrightarrow{\chi} a_{i+2}) = w(b_i) + w(b_{i+1})$
- We can ignore a_{i+2} and b_{i+2} and only build the following
  - $b_i \xrightarrow{\chi} a_{i+1} \xrightarrow{\lambda} b_{i+1} \xrightarrow{\chi}$
- Minimum reverse weight $w_{rev}(a_{i+1})$: minimum over the weight of all compatible b_i's
- We introduced a new method to compute the $w_{rev}(a_{i+1}) \rightarrow$ ignore b_i
- 2-round trail core: $\xrightarrow{\chi} a_{i+1} \xrightarrow{\lambda} b_{i+1} \xrightarrow{\chi}$

Generate 2-round trail cores up to weight T_2

$$\xrightarrow{\chi} a_1 \xrightarrow{\lambda} b_1 \xrightarrow{\chi}$$

- Generate all differences at a_1 with $w(Q_2)$ up to T_2

Generate 2-round trail cores up to weight T_2

$$\xrightarrow{\chi} a_1 \xrightarrow{\lambda} b_1 \xrightarrow{\chi}$$

- Generate all differences at a_1 with $w(Q_2)$ up to T_2
- Compute b_1

Generate 2-round trail cores up to weight T_2

$$\xrightarrow{\chi} a_1 \xrightarrow{\lambda} b_1 \xrightarrow{\chi}$$

- Generate all differences at a_1 with $w(Q_2)$ up to T_2
- Compute $b_1 = \lambda(a_1)$

Generate 2-round trail cores up to weight T_2

$$\xrightarrow{\chi} a_1 \xrightarrow{\lambda} b_1 \xrightarrow{\chi}$$

- Generate all differences at a_1 with $w(Q_2)$ up to T_2
- Compute $b_1 = \lambda(a_1)$
- If $w_{rev}(a_1) + w(b_1) \leq T_2$ then (a_1, b_1) is a valid pair

Generate 2-round trail cores up to weight T_2

$$\xrightarrow{\chi} a_1 \xrightarrow{\lambda} b_1 \xrightarrow{\chi}$$

- Generate all differences at a_1 with $w(Q_2)$ up to T_2
- Compute $b_1 = \lambda(a_1)$
- If $w_{rev}(a_1) + w(b_1) \leq T_2$ then (a_1, b_1) is a valid pair
- How to scan the space of all 2-round trail cores up to T_2?

Generate 2-round trail cores up to weight T_2

$$\xrightarrow{\chi} a_1 \xrightarrow{\lambda} b_1 \xrightarrow{\chi}$$

- Generate all differences at a_1 with $w(Q_2)$ up to T_2
- Compute $b_1 = \lambda(a_1)$
- If $w_{rev}(a_1) + w(b_1) \leq T_2$ then (a_1, b_1) is a valid pair
- How to scan the space of all 2-round trail cores up to T_2?
- Tree search approach [Mella, Daemen, Van Assche, ToSC 2016]

Tree search

$$\xrightarrow{\chi} a_1 \xrightarrow{\lambda} b_1 \xrightarrow{\chi}$$

- Each node represents an ordered list of active bits at a_1 and corresponding b_1

Tree search

$$\xrightarrow{\chi} a_1 \xrightarrow{\lambda} b_1 \xrightarrow{\chi}$$

- Each node represents an ordered list of active bits at a_1 and corresponding b_1
- Three functions to visit all nodes of a tree

Tree search

$$\xrightarrow{\chi} a_1 \xrightarrow{\lambda} b_1 \xrightarrow{\chi}$$

- Each node represents an ordered list of active bits at a_1 and corresponding b_1
- Three functions to visit all nodes of a tree
  - toChild(): adds a new active bit to a_1

Tree search

$$\xrightarrow{\chi} a_1 \xrightarrow{\lambda} b_1 \xrightarrow{\chi}$$

- Each node represents an ordered list of active bits at a_1 and corresponding b_1
- Three functions to visit all nodes of a tree
  - toChild(): adds a new active bit to a_1
  - toSibling(): Changes the last added active bit of a_1

$$\xrightarrow{\chi} a_1 \xrightarrow{\lambda} b_1 \xrightarrow{\chi}$$

- Each node represents an ordered list of active bits at a_1 and corresponding b_1
- Three functions to visit all nodes of a tree
  - toChild(): adds a new active bit to a_1
  - toSibling(): Changes the last added active bit of a_1
  - toParent(): deletes the last added active bit of a_1

Tree search

$$\xrightarrow{\chi} a_1 \xrightarrow{\lambda} b_1 \xrightarrow{\chi}$$

- Each node represents an ordered list of active bits at a_1 and corresponding b_1
- Three functions to visit all nodes of a tree
  - toChild(): adds a new active bit to a_1
  - toSibling(): Changes the last added active bit of a_1
  - toParent(): deletes the last added active bit of a_1
- **Score**: a lower bound on the weight of a node and all its descendants

Tree search

$$\xrightarrow{\chi} a_1 \xrightarrow{\lambda} b_1 \xrightarrow{\chi}$$

- Each node represents an ordered list of active bits at a_1 and corresponding b_1
- Three functions to visit all nodes of a tree
  - toChild(): adds a new active bit to a_1
  - toSibling(): Changes the last added active bit of a_1
  - toParent(): deletes the last added active bit of a_1
- **Score**: a lower bound on the weight of a node and all its descendants
- Score allows to prune the tree

Tree search

$$\xrightarrow{\chi} a_1 \xrightarrow{\lambda} b_1 \xrightarrow{\chi}$$

- Each node represents an ordered list of active bits at a_1 and corresponding b_1
- Three functions to visit all nodes of a tree
  - toChild(): adds a new active bit to a_1
  - toSibling(): Changes the last added active bit of a_1
  - toParent(): deletes the last added active bit of a_1
- **Score**: a lower bound on the weight of a node and all its descendants
- Score allows to prune the tree $\rightarrow$ faster scan (larger space)

Tree search

$$\xrightarrow{\chi} a_1 \xrightarrow{\lambda} b_1 \xrightarrow{\chi}$$

- Each node represents an ordered list of active bits at a_1 and corresponding b_1
- Three functions to visit all nodes of a tree
  - toChild(): adds a new active bit to a_1
  - toSibling(): Changes the last added active bit of a_1
  - toParent(): deletes the last added active bit of a_1
- **Score**: a lower bound on the weight of a node and all its descendants
- Score allows to prune the tree $\rightarrow$ faster scan (larger space)
- Score can be defined separately for a_1 and b_1

Tree search

$$\xrightarrow{\chi} a_1 \xrightarrow{\lambda} b_1 \xrightarrow{\chi}$$

- Each node represents an ordered list of active bits at a_1 and corresponding b_1
- Three functions to visit all nodes of a tree
  - toChild(): adds a new active bit to a_1
  - toSibling(): Changes the last added active bit of a_1
  - toParent(): deletes the last added active bit of a_1
- **Score**: a lower bound on the weight of a node and all its descendants
- Score allows to prune the tree $\rightarrow$ faster scan (larger space)
- Score can be defined separately for a_1 and b_1
- We introduced a new generic method to compute the score at a_1

Tree search

$$\xrightarrow{\chi} a_1 \xrightarrow{\lambda} b_1 \xrightarrow{\chi}$$

- Each node represents an ordered list of active bits at a_1 and corresponding b_1
- Three functions to visit all nodes of a tree
  - toChild(): adds a new active bit to a_1
  - toSibling(): Changes the last added active bit of a_1
  - toParent(): deletes the last added active bit of a_1
- **Score**: a lower bound on the weight of a node and all its descendants
- Score allows to prune the tree $\rightarrow$ faster scan (larger space)
- Score can be defined separately for a_1 and b_1
- We introduced a new generic method to compute the score at a_1
- ... and a new method to compute the score at b_1 for SUBTERRANEAN

$$\boxed{\chi} \longrightarrow \boxed{\lambda} \longrightarrow \boxed{\chi}$$

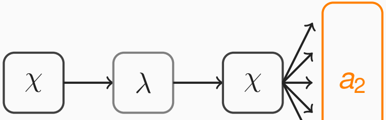
$$\xrightarrow{\chi} a_1 \xrightarrow{\lambda} b_1 \xrightarrow{\chi}$$

Trail extension

$$\boxed{\chi} \longrightarrow \boxed{\lambda} \longrightarrow \boxed{\chi}$$

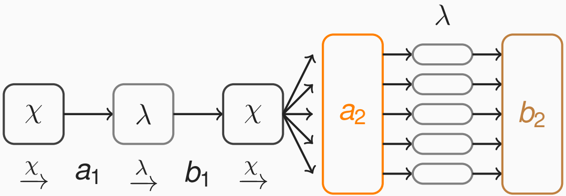
$$\xrightarrow{\chi} a_1 \xrightarrow{\lambda} b_1 \xrightarrow{\chi}$$

- Forward extension



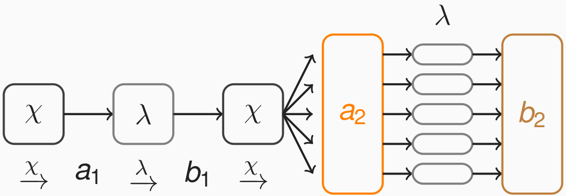
$\xrightarrow{\chi} a_1 \xrightarrow{\lambda} b_1 \xrightarrow{\chi}$

- Forward extension
    - Generating all possible a_2's



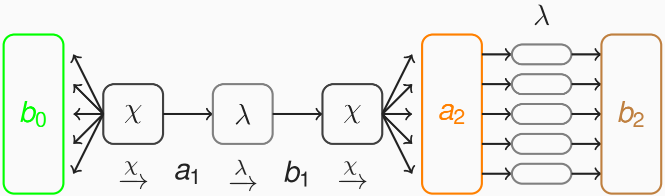
- Forward extension
  - Generating all possible a_2's
  - Computing the corresponding b_2 for each a_2

Trail extension



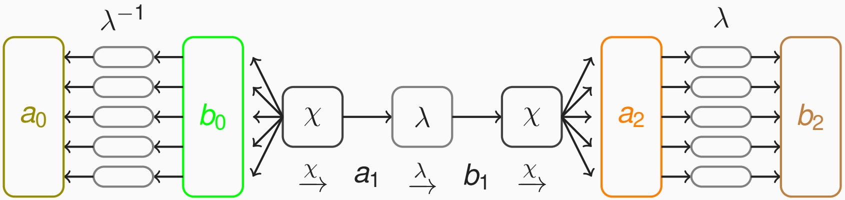
- Forward extension
  - Generating all possible a_2's
  - Computing the corresponding b_2 for each a_2
- Backward extension

Trail extension



- Forward extension
  - Generating all possible a_2's
  - Computing the corresponding b_2 for each a_2
- Backward extension
  - We introduced a new method to efficiently generate all possible b_0's

Trail extension



- Forward extension
  - Generating all possible a_2's
  - Computing the corresponding b_2 for each a_2
- Backward extension
  - We introduced a new method to efficiently generate all possible b_0's
  - Computing the corresponding a_0 for each b_0

Application

FPGA benchmarking: NIST lightweight round 2 candidates

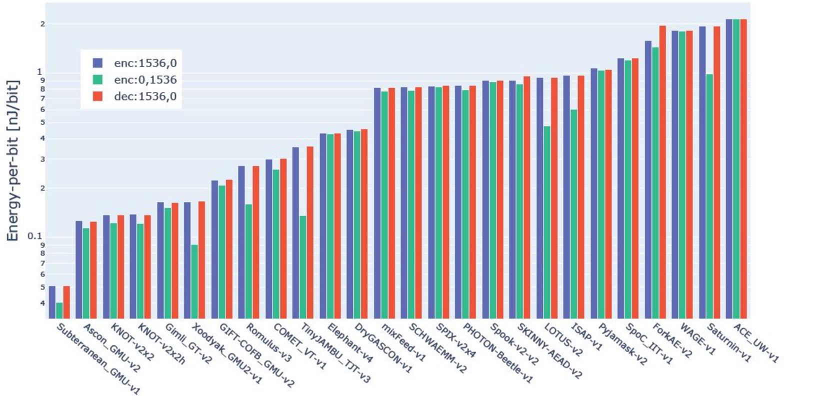


Figure 35: Energy-per-bit for Authenticated Encryption and Decryption of 1536-Byte messages at 75MHz

[Mohajerani et al. 2021]

FPGA benchmarking: NIST lightweight round 2 candidates

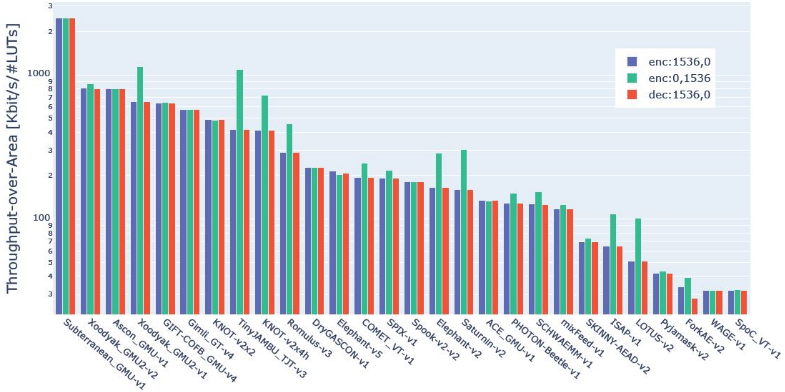
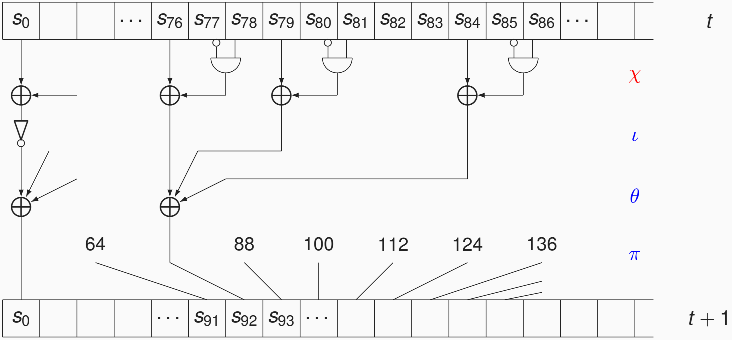


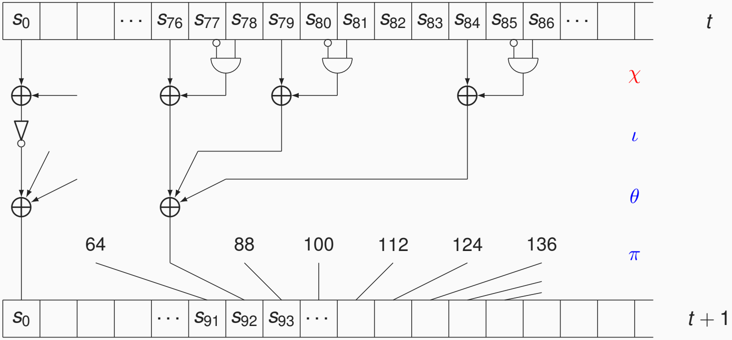
Figure 36: Throughput-over-Area for Authenticated Encryption and Decryption of 1536-Byte messages at 75MHz

[Mohajerani et al. 2021]

The SUBTERRANEAN 2.0 round function

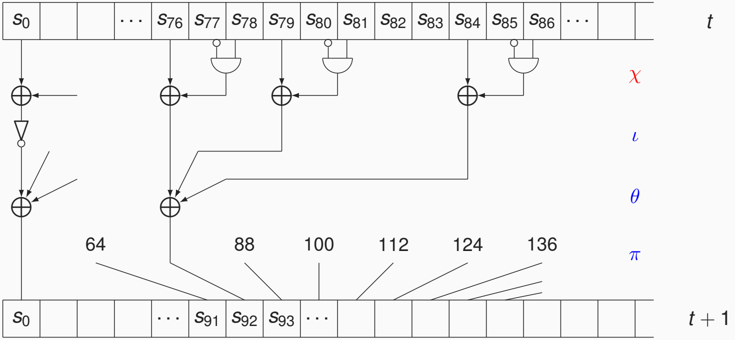


The SUBTERRANEAN 2.0 round function



$$\chi : \quad s_i \leftarrow \quad s_i + (s_{i+1} + 1) s_{i+2} \; .$$

The SUBTERRANEAN 2.0 round function



$$\chi : \quad s_i \leftarrow \quad s_i + (s_{i+1} + 1)s_{i+2} \; .$$

$$\iota : \quad s_i \leftarrow \quad s_i + \delta_i \; ,$$

$$\theta : \quad s_i \leftarrow \quad s_i + s_{i+3} + s_{i+8} \; ,$$

$$\pi : \quad s_i \leftarrow \quad s_{12i} \; .$$

Lower bound on the weight of trails in SUBTERRANEAN

Goal: scanning 8-round trail cores in SUBTERRANEAN

- 2-round trail cores up to 28

Lower bound on the weight of trails in SUBTERRANEAN

Goal: scanning 8-round trail cores in SUBTERRANEAN

- 2-round trail cores up to 28
- 3-round trail cores up to 40

Lower bound on the weight of trails in SUBTERRANEAN

Goal: scanning 8-round trail cores in SUBTERRANEAN

- 2-round trail cores up to 28
- 3-round trail cores up to 40
- 4-round trail cores up to 57 $\rightarrow$ no trail core

Lower bound on the weight of trails in SUBTERRANEAN

Goal: scanning 8-round trail cores in SUBTERRANEAN

- 2-round trail cores up to 28
- 3-round trail cores up to 40
- 4-round trail cores up to 57 $\rightarrow$ no trail core
- The best 4-round trail core we found has weight 58

Lower bound on the weight of trails in SUBTERRANEAN

Goal: scanning 8-round trail cores in SUBTERRANEAN

- 2-round trail cores up to 28
- 3-round trail cores up to 40
- 4-round trail cores up to 57 $\rightarrow$ no trail core
- The best 4-round trail core we found has weight 58
- In the worst case, two 4-rounds with weight 58 can be compatible and form an 8-round trail core

Lower bound on the weight of trails in SUBTERRANEAN

Goal: scanning 8-round trail cores in SUBTERRANEAN

- 2-round trail cores up to 28
- 3-round trail cores up to 40
- 4-round trail cores up to 57 $\rightarrow$ no trail core
- The best 4-round trail core we found has weight 58
- In the worst case, two 4-rounds with weight 58 can be compatible and form an 8-round trail core
- $w(Q_8) = w(Q_4) + w(Q'_4) \geq 58 + 58 = 116$

Lower bound on the weight of trails in SUBTERRANEAN

Goal: scanning 8-round trail cores in SUBTERRANEAN

- 2-round trail cores up to 28
- 3-round trail cores up to 40
- 4-round trail cores up to 57 $\rightarrow$ no trail core
- The best 4-round trail core we found has weight 58
- In the worst case, two 4-rounds with weight 58 can be compatible and form an 8-round trail core
- $w(Q_8) = w(Q_4) + w(Q'_4) \geq 58 + 58 = 116$

# rounds:	1	2	3	4	5	6	7	8
lower bound (this work):	2	8	25	58	≥ 62	≥ 78	≥ 80	≥ 116
lower bound [DMMR20]:	2	8	25	$[49, 58]$	≥ 54	≥ 65	≥ 70	≥ 98

Lower bound on the weight of differential trail cores

Conclusion

Conclusion

- We introduced:

Conclusion

- We introduced:
  - a **score function** that lower bounds the weight of a 2-round trail core and all its descendants during the tree search

Conclusion

- We introduced:
  - a **score function** that lower bounds the weight of a 2-round trail core and all its descendants during the tree search
  - a method to **efficiently compute the minimum weight of a trail core**

Conclusion

- We introduced:
  - a **score function** that lower bounds the weight of a 2-round trail core and all its descendants during the tree search
  - a method to **efficiently compute the minimum weight of a trail core**
  - a method to **efficiently perform backward extension**

Conclusion

- We introduced:
  - a **score function** that lower bounds the weight of a 2-round trail core and all its descendants during the tree search
  - a method to **efficiently compute the minimum weight of a trail core**
  - a method to **efficiently perform backward extension**
  - a software tool for **differential trail search tailored for** SUBTERRANEAN

Conclusion

- We introduced:
  - a **score function** that lower bounds the weight of a 2-round trail core and all its descendants during the tree search
  - a method to **efficiently compute the minimum weight of a trail core**
  - a method to **efficiently perform backward extension**
  - a software tool for **differential trail search tailored for** SUBTERRANEAN $\rightarrow$ **new and more precise lower bounds**

Conclusion

- We introduced:
  - a **score function** that lower bounds the weight of a 2-round trail core and all its descendants during the tree search
  - a method to **efficiently compute the minimum weight of a trail core**
  - a method to **efficiently perform backward extension**
  - a software tool for **differential trail search tailored for** SUBTERRANEAN $\rightarrow$ **new and more precise lower bounds**
- Tree search is a very strong and useful tool

Conclusion

- We introduced:
  - a **score function** that lower bounds the weight of a 2-round trail core and all its descendants during the tree search
  - a method to **efficiently compute the minimum weight of a trail core**
  - a method to **efficiently perform backward extension**
  - a software tool for **differential trail search tailored for** SUBTERRANEAN $\rightarrow$ **new and more precise lower bounds**
- Tree search is a very strong and useful tool

Thanks for your attention!