Practical Near-Collisions and Collisions on Reduced-Round ECHO-256 Compression Function

Jérémy Jean and Pierre-Alain Fouque

Ecole Normale Supérieure

FSE'2011

February 14, 2011

Outline of the talk

Outline

- Previous cryptanalysis
- Description of ECHO-256
- Collision attack on 4-round ECHO-256
- Rebound attacks and improvements

Previous cryptanalysis of ECHO-256

Hash function

Rounds	Time	Memory	Type	Reference
5/8	2^{112}	$2^{85.3}$	collision	[Schläffer-eprint10]

Compression function

Rounds	Time	Memory	Type	Reference
3/8	2^{64}	2^{32}	free-start collision	[Peyrin-C10]
3/8	2^{96}	2^{32}	semi-free-start collision	[Peyrin-C10]
4.5/8	2^{96}	2^{32}	distinguisher	[Peyrin-C10]
4/8	$\mathbf{2^{36}}$	$\mathbf{2^{16}}$	**distinguisher**	**new**
4/8	$\mathbf{2^{52}}$	$\mathbf{2^{16}}$	**semi-free-start collision**	**new**
6/8	2^{160}	2^{128}	collision, chosen salt	[Schläffer-eprint10]
7/8	2^{160}	2^{128}	distinguisher, chosen salt	[Schläffer-eprint10]

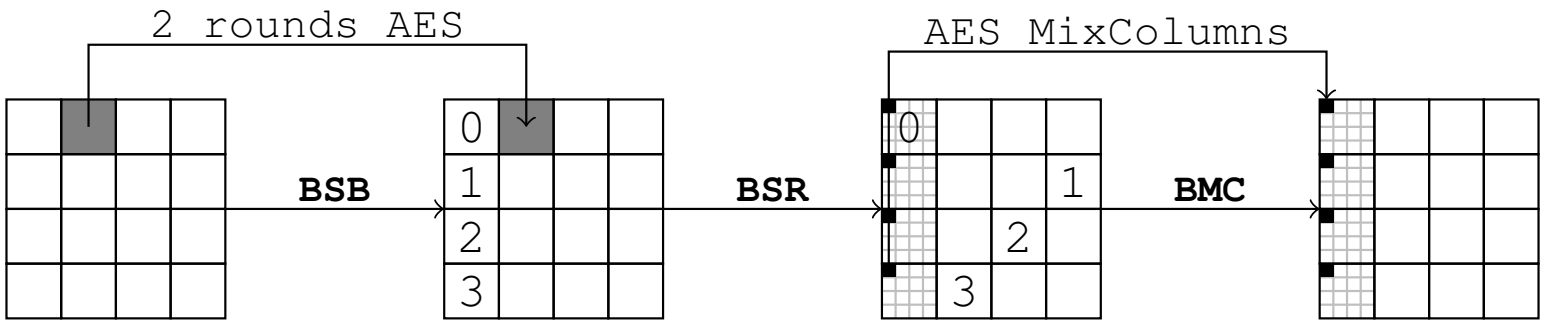
Permutation

Rounds	Time	Memory	Type	Reference
8/8	2^{182}	2^{37}	distinguisher	[SLWSO-A10]
8/8	2^{151}	2^{67}	distinguisher	[NayaPlasencia-eprint10]

Description of the hash function

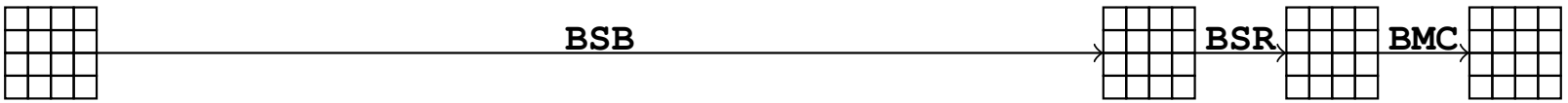
ECHO-256

- Merkle-Damgård construction
- HAIFA design (counter & salt)
- 2048-bit internal state as a 4×4 matrix of AES states
- 8-round AES-based permutation : BSB, BSR, BMC
- Output transformation : compress and truncate



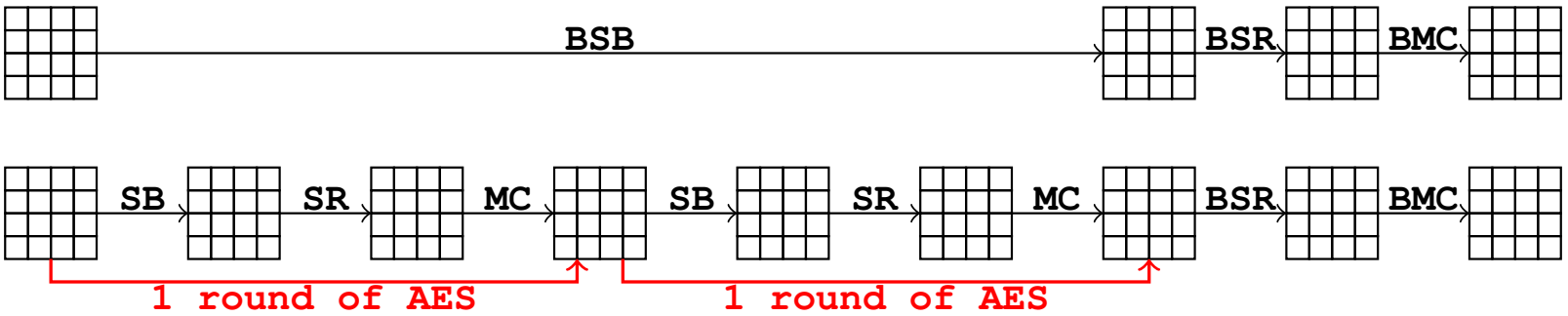
Alternative view

- Breaking down to the AES-state level of operations
  - **SuperSBox** = **SB – MC – SB** [LMRRS-A09, GP-FSE10]
  - **SuperMixColumns** = **MC – BMC** [Schläffer-SAC10]



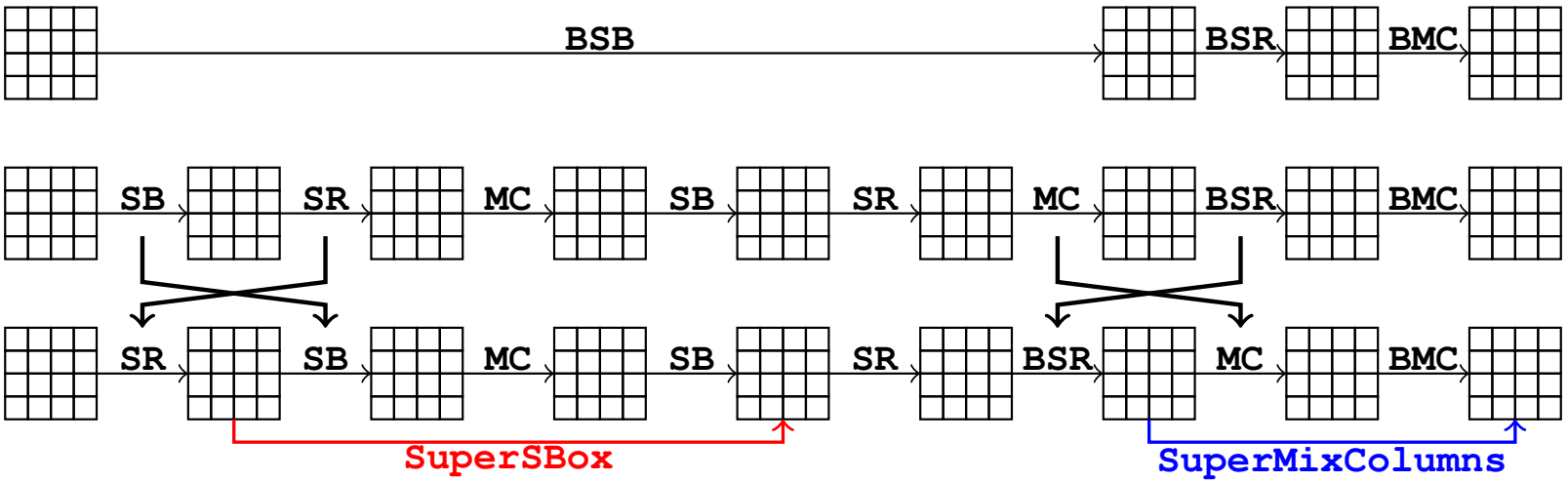
Alternative view

- Breaking down to the AES-state level of operations
  - **SuperSBox** = **SB – MC – SB** [LMRRS-A09, GP-FSE10]
  - **SuperMixColumns** = **MC – BMC** [Schläffer-SAC10]



Alternative view

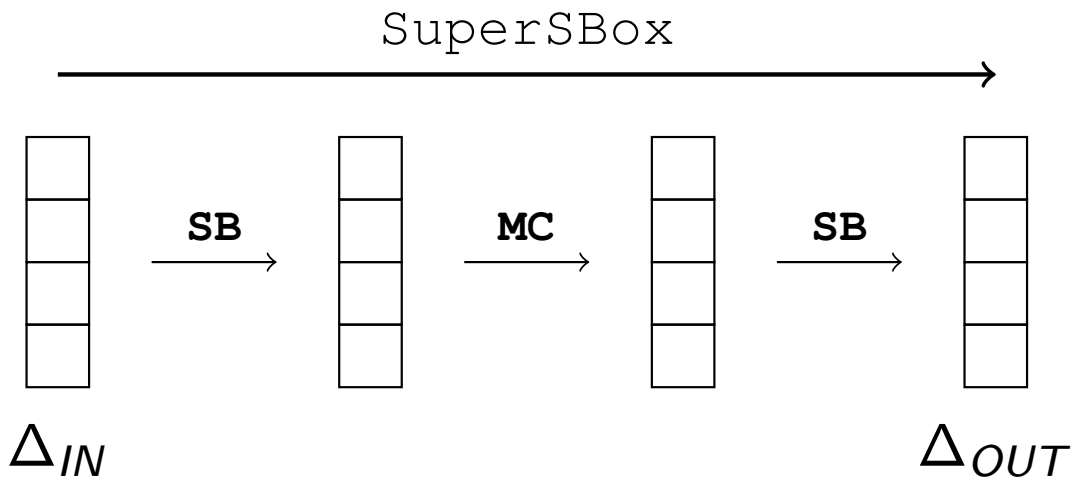
- Breaking down to the AES-state level of operations
  - **SuperSBox** = **SB – MC – SB** [LMRRS-A09, GP-FSE10]
  - **SuperMixColumns** = **MC – BMC** [Schläffer-SAC10]



SuperSBox

Description

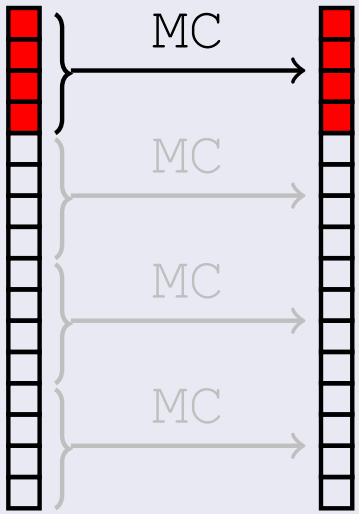
- Super transformation used in [LMRRS-A09, GP-FSE10]
- **SuperSBox = SB – MC – SB**
- Works on 32-bit AES-columns
- $\mathbb{P}(\Delta_{IN} \rightarrow \Delta_{OUT}$ exists$) \approx 1/2$



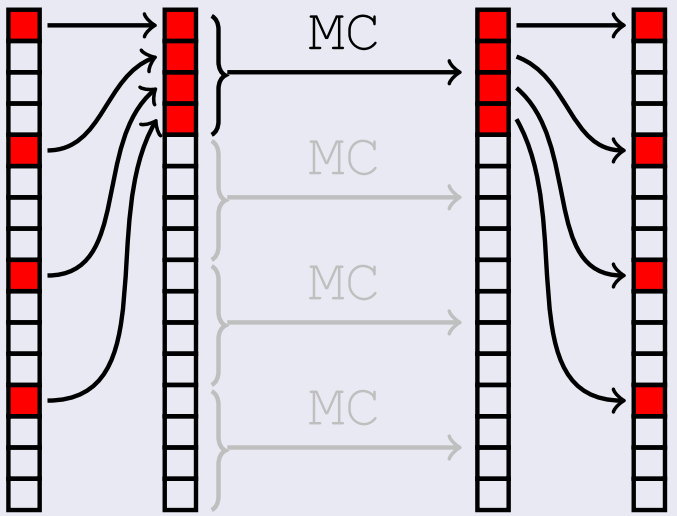
MixColumns and BigMixColumns

4 parallel applications of MixColumns/BigMixColumns

MixColumns



BigMixColumns

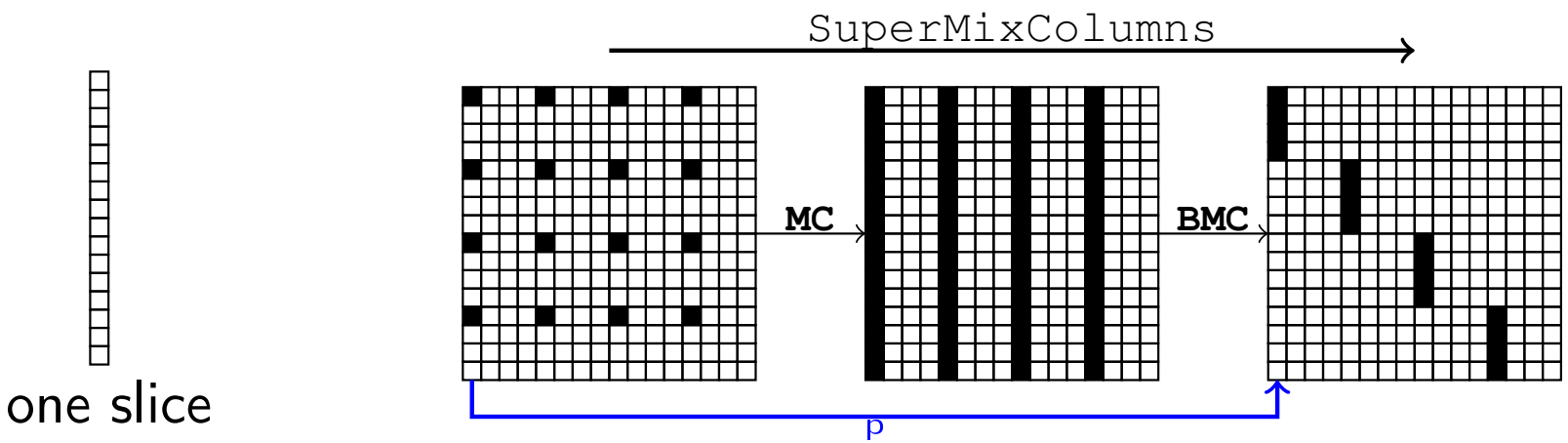


MC : AES MixColumns

SuperMixColumns

16×16 matrix of SMC

- Super transformation introduced in [Schläffer-SAC10]
- Works on 16×1 byte-slices
- $\mathbf{M}_{SMC} = \mathbf{M} \otimes \mathbf{M}$ ($\mathbf{M}$ from **MixColumns**)
- Branch number $= 8$ (optimal : 17)
- Sparse paths : $4 \rightarrow 16 \rightarrow 4$, $p = 2^{-24}$



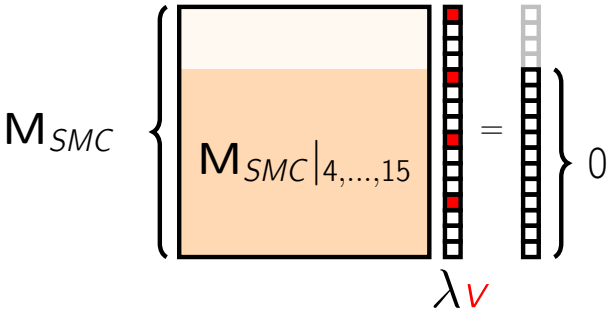
SuperMixColumns

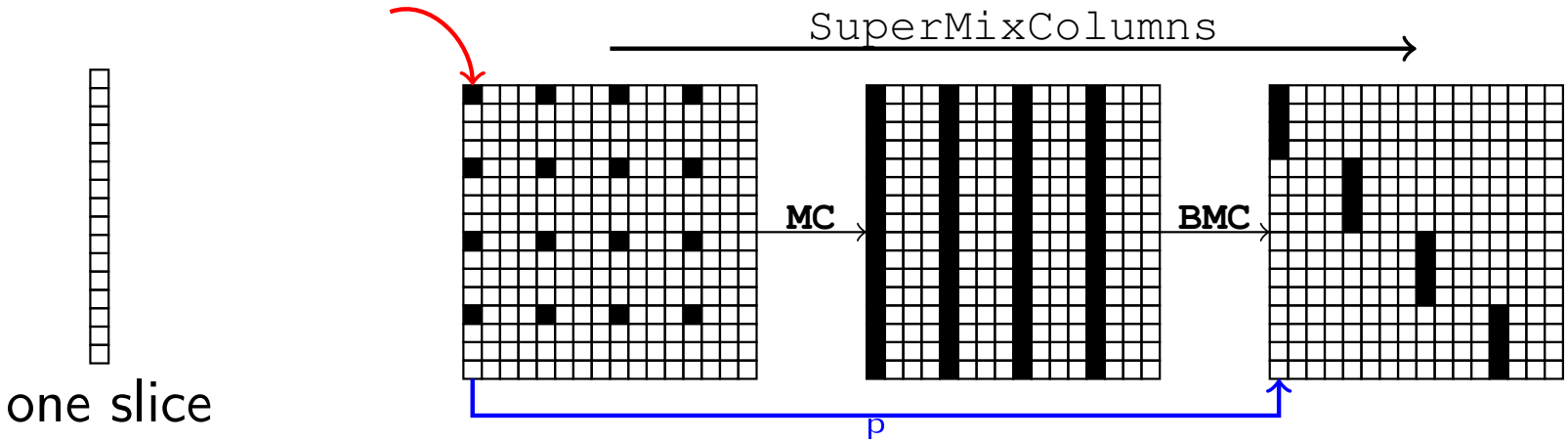
Restriction

Sparse paths $\Longrightarrow$ one-dimensional subsets of kernels

$$\mathrm{Span}(v) \subset \ker\left(\mathbf{M}_{SMC}|_{4,\ldots,15}\right)$$

$$v = [14\ 0\ 0\ 0\ 9\ 0\ 0\ 0\ 13\ 0\ 0\ 0\ 11\ 0\ 0\ 0]^{\mathrm{T}}$$



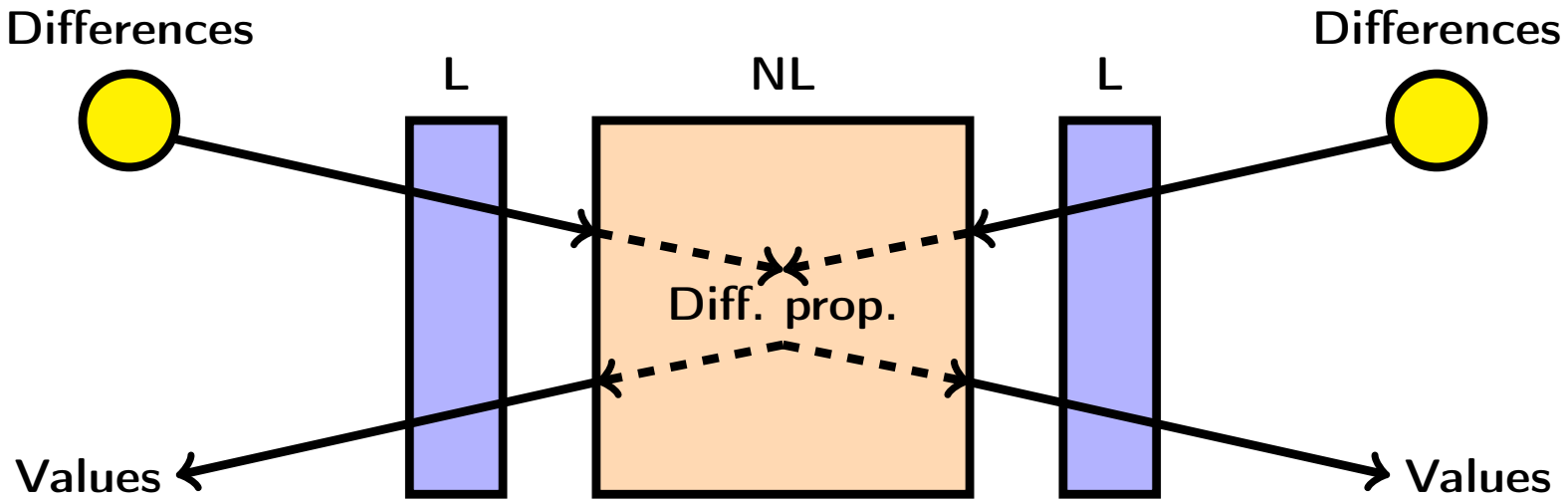


Rebound technique

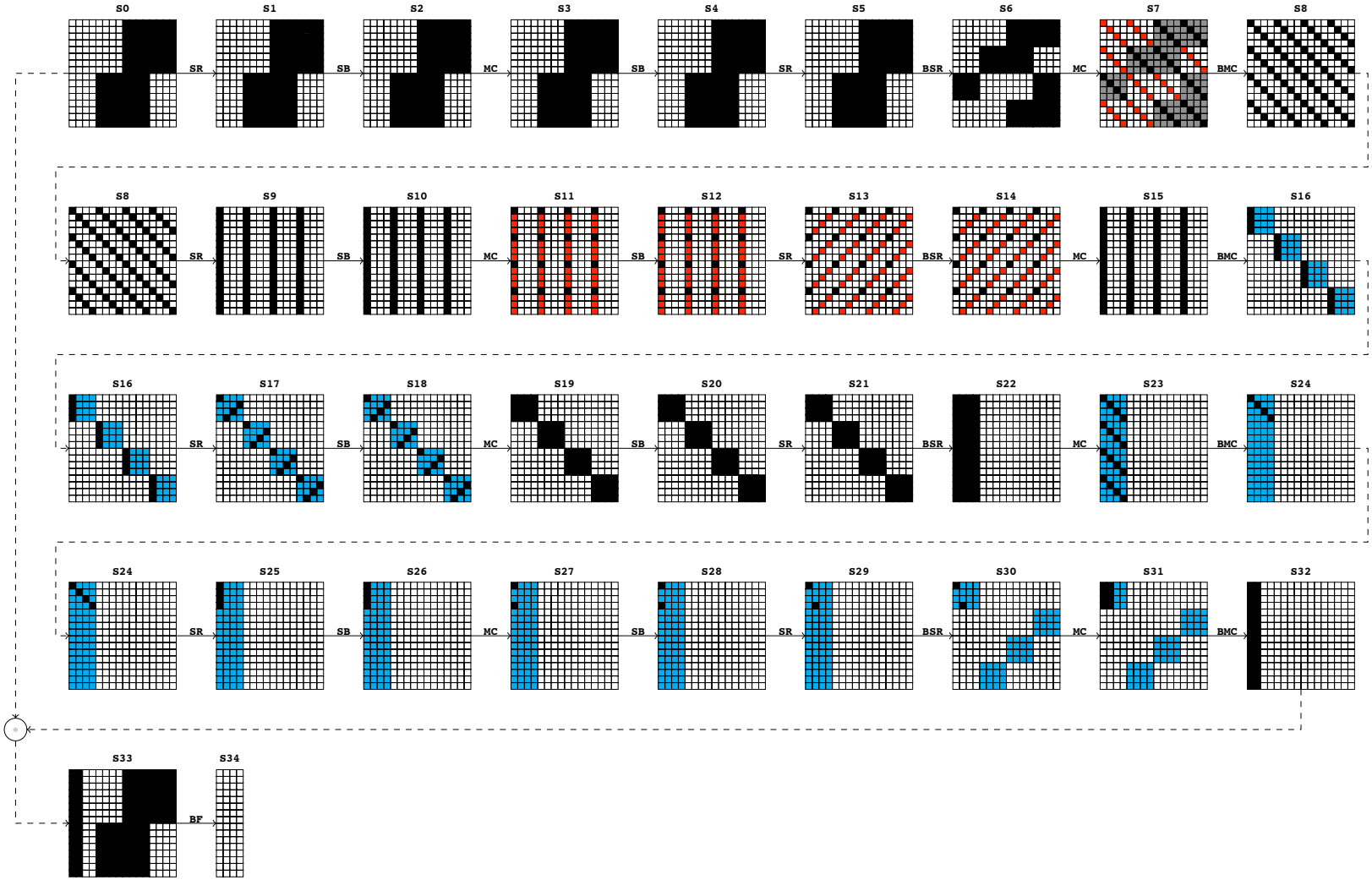
For a given truncated differential path

Set differences and values around a non-linear layer using its differential properties with amortized complexity one

NL = AES SBox or SuperSBox



Path for the 4-round collision attack



Finding a valid pair

Path

- Path from [Schläffer-SAC10]
- Modified in the first round

Overview

- Differential attack
- Two subparts solved *sequentially*
- One merging step

Finding a valid pair

Path

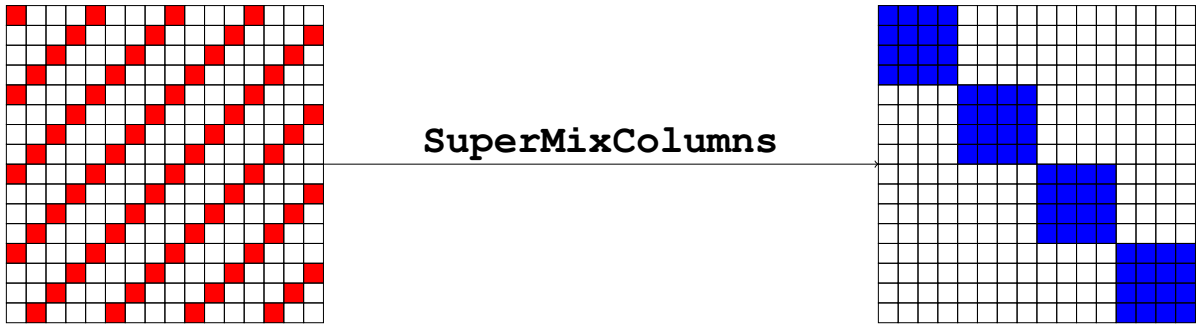
- Path from [Schläffer-SAC10]
- Modified in the first round

Overview

- Differential attack
- Two subparts solved *sequentially*
- One merging step

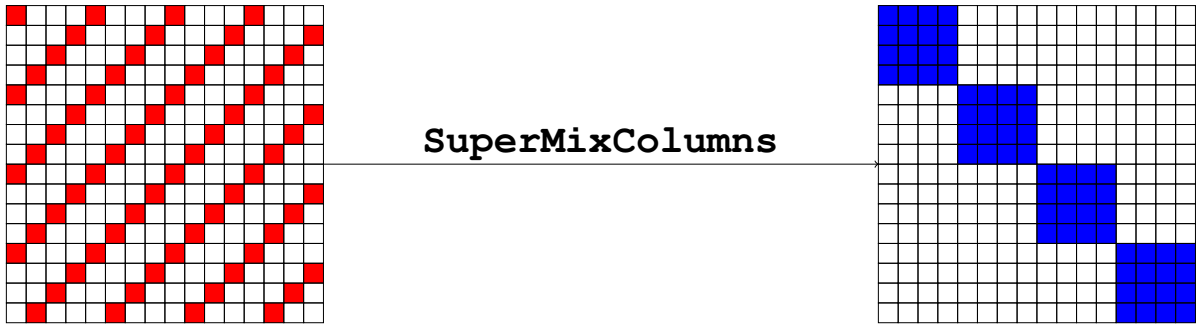
Merging the two subparts

Goal : find values for white bytes.
Known : red bytes (1st subpart), blue bytes (2nd subpart)



Merging the two subparts

Goal : find values for white bytes.
Known : red bytes (1st subpart), blue bytes (2nd subpart)

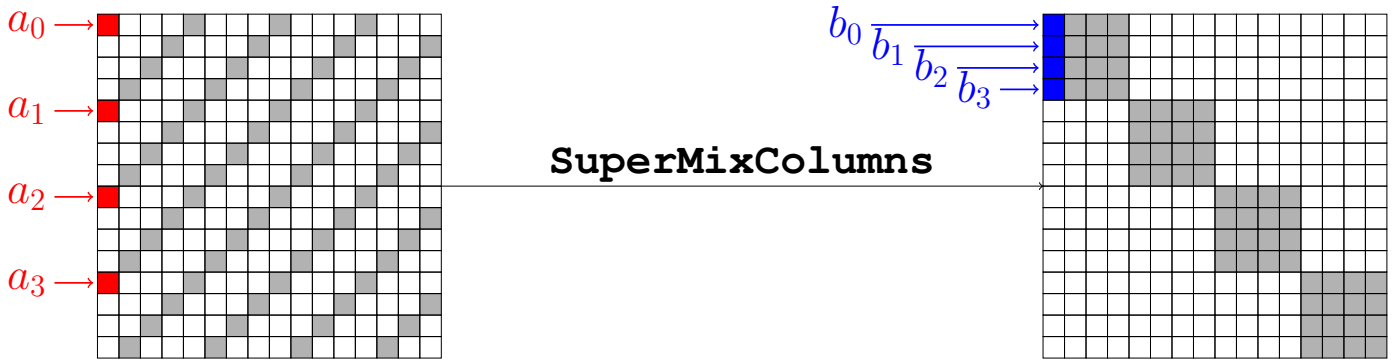


Problem

- Fails w.h.p in [Schläffer-SAC10]
- Cause : system without solution

Merging the two subparts

Goal : find values for white bytes.
Known : red bytes (1st subpart), blue bytes (2nd subpart)



Problem

- Fails w.h.p in [Schläffer-SAC10]
- Cause : system without solution

Correction

- Merge still possible...
- ...but 128-bit constraint

$$\text{slice\#0} : 2a_0 + 3a_1 + a_2 + a_3 = 14b_0 + 11b_1 + 13b_2 + 9b_3$$

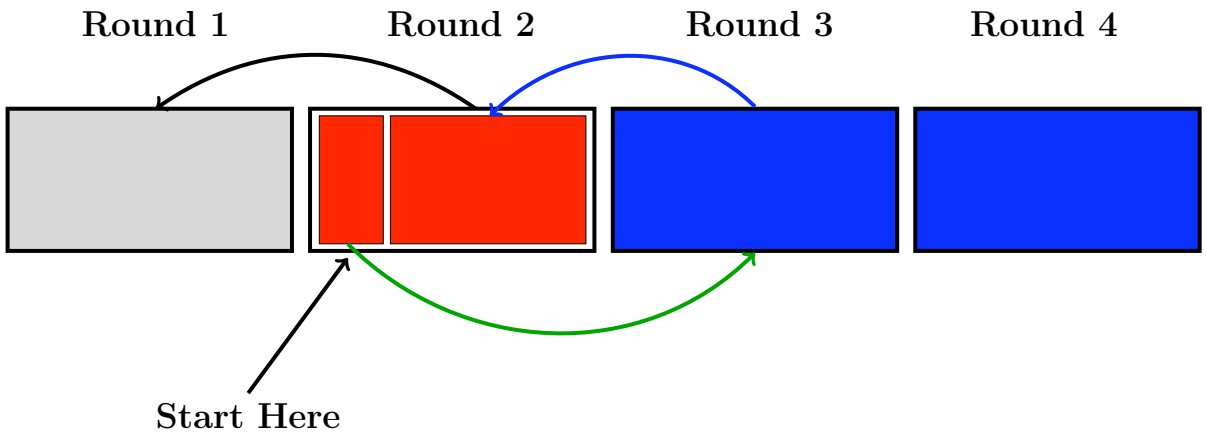
meet-in-the-middle condition

How to solve the merge problem

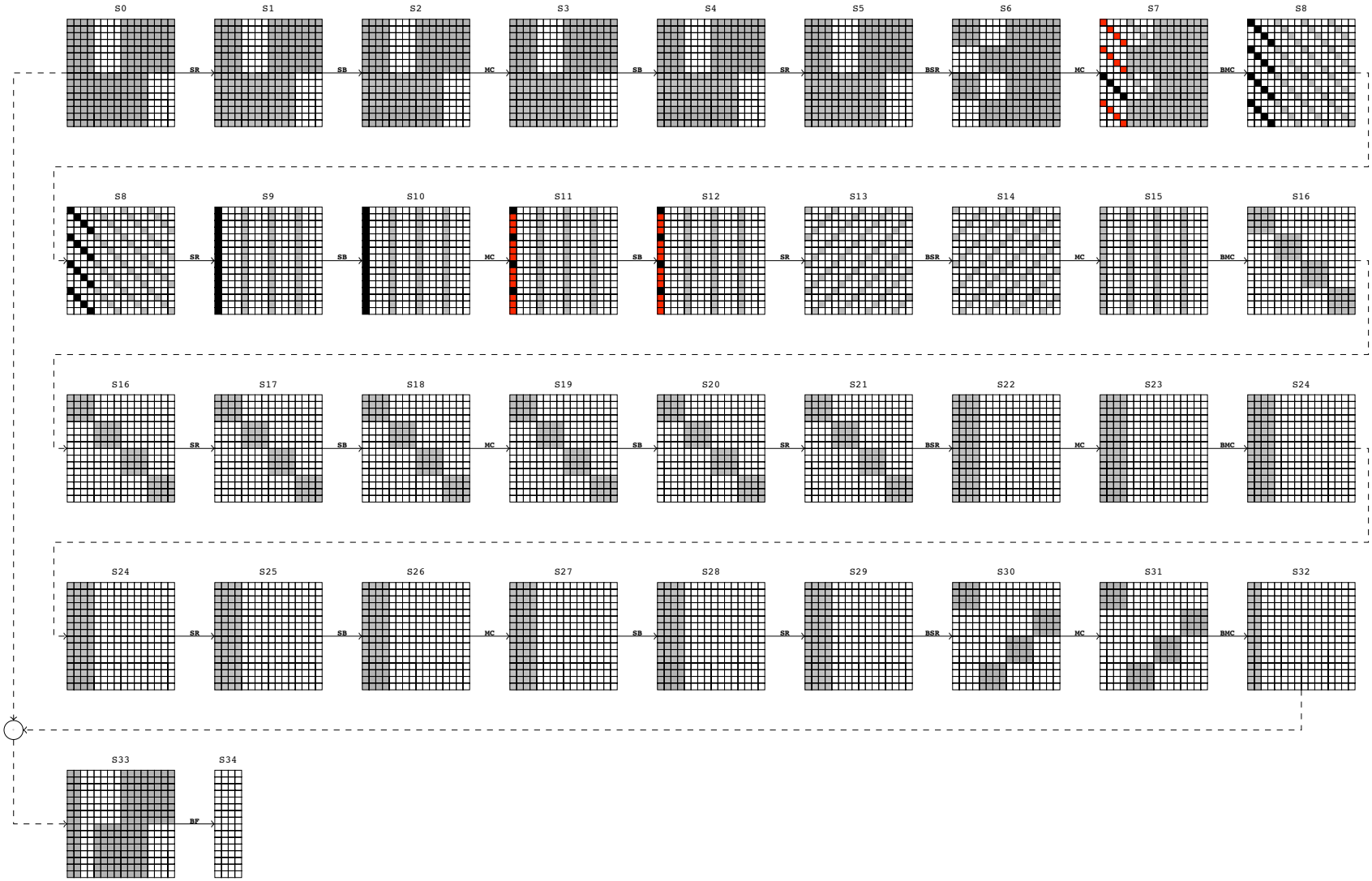
Steps of the attack

1. Solve first column of Round 2
2. Fix differences for a valid path through SMC
3. Input Differences $\Longrightarrow$ Values for Rounds 3 & 4
4. Deduce three last columns
5. Values for Round 1 $\Longrightarrow$ Collision

First subpart Second subpart



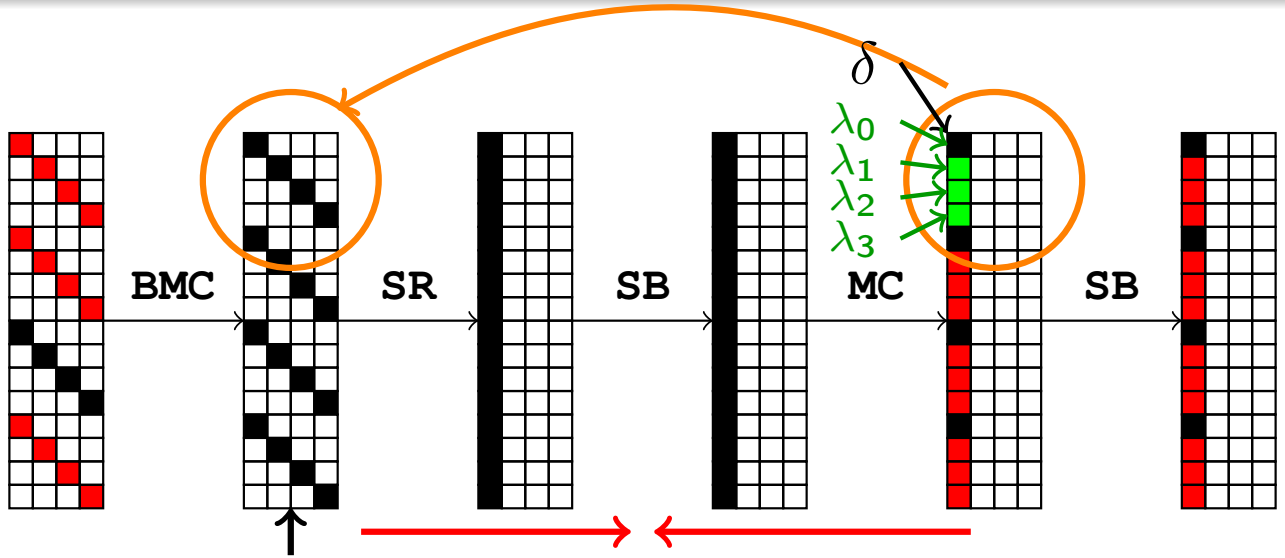
Step 1


S0
S1
S2
S3
S4
S5
S6
S7
S8
SR
SB
MC
BSR
BMC
S8
S9
S10
S11
S12
S13
S14
S15
S16
S16
S17
S18
S19
S20
S21
S22
S23
S24
S24
S25
S26
S27
S28
S29
S30
S31
S32
S33
S34
BF

Step 1

Method

1. Randomize δ, λ_0, λ_1, λ_2, λ_3
2. Propagate values backwards
3. Linearly deduce all differences
4. Rebound technique on the AES SBox to finish 2^{12}

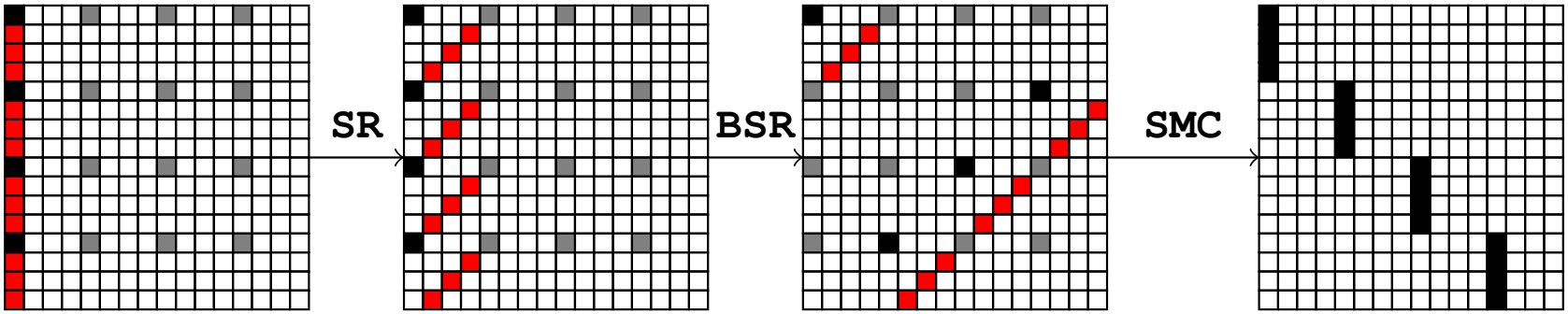


Step 1 → Step 2

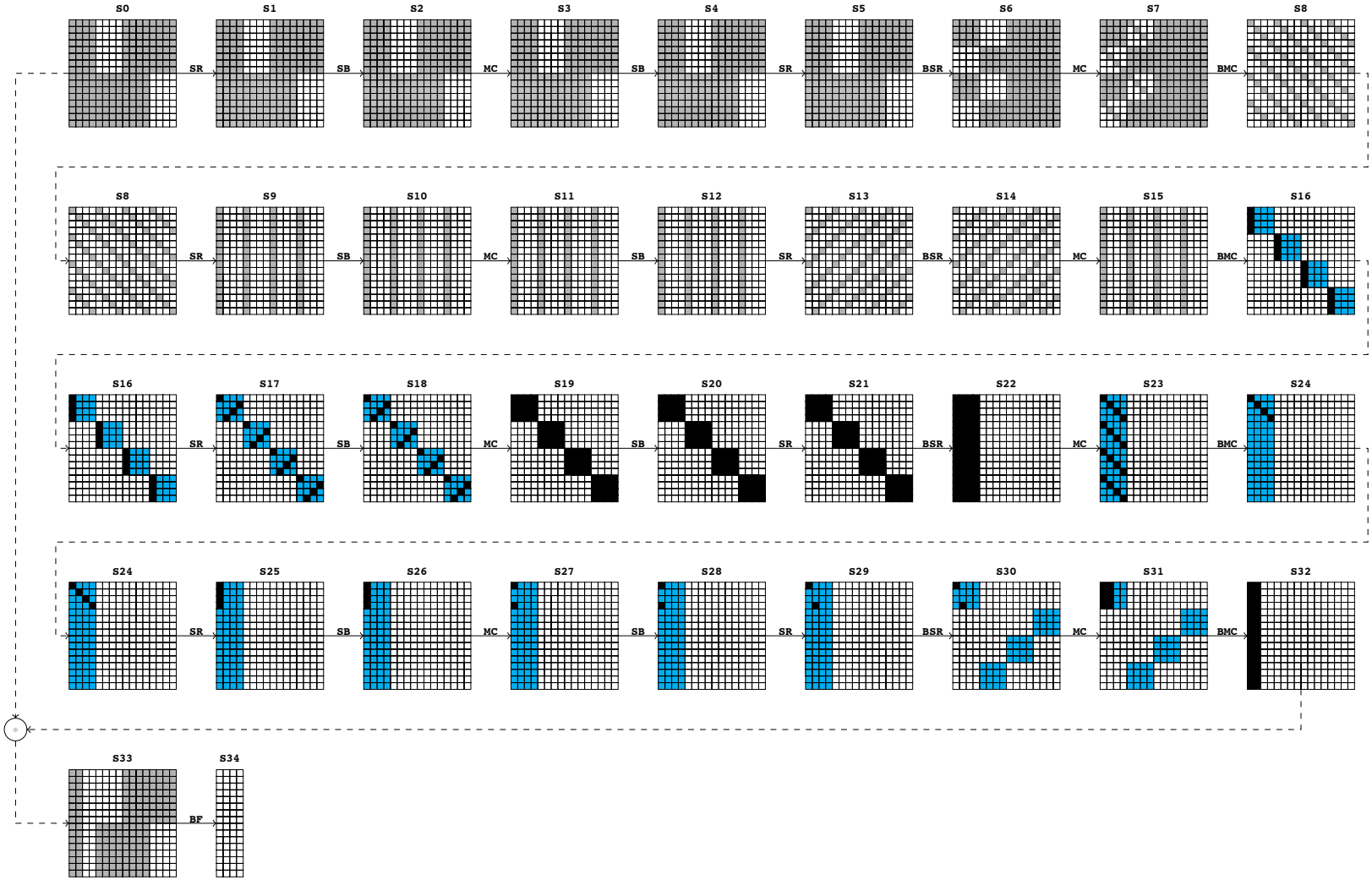
Known : Red values and black differences (from 1st column)

Differences

- One difference per column is known after BSR ■
- Sparse SMC path ⟹ fix all differences after SMC ■



Step 2 : pair for the second subpart

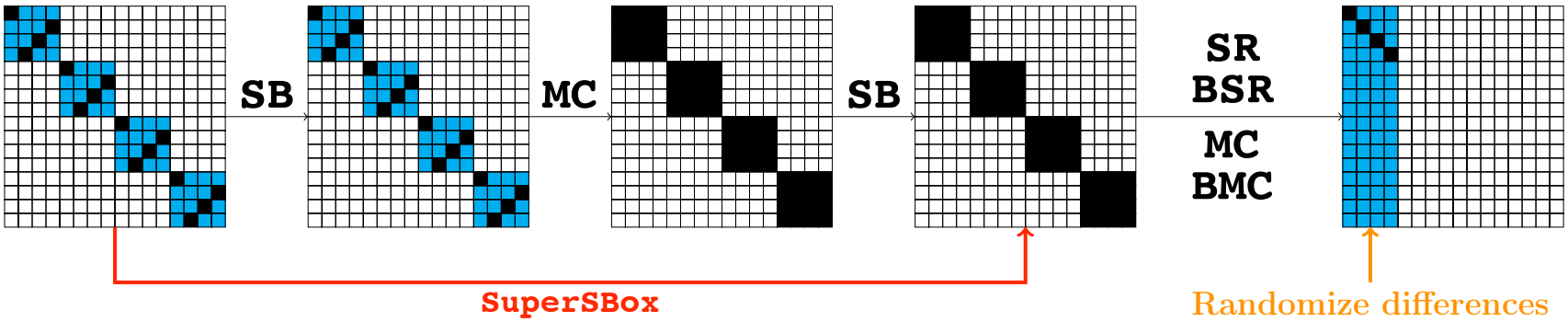


Step 2 : pair for the second subpart

Goal : Find values for blue bytes
Known : Input differences (Black bytes in the first state)

Method

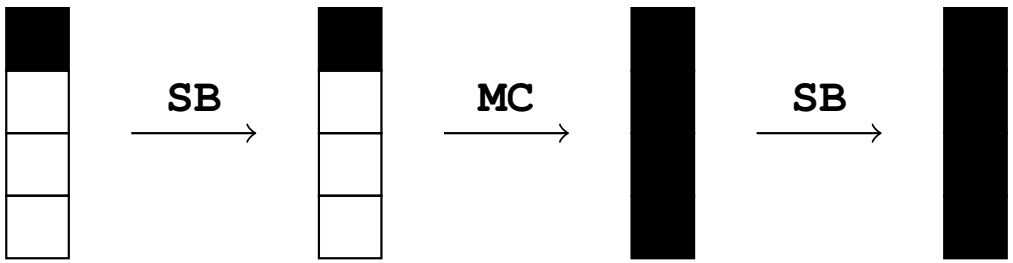
- Randomize 4 differences
- Linearly learn all output differences
- Find input values for 16 SuperSBoxes $p = 2^{-16}$



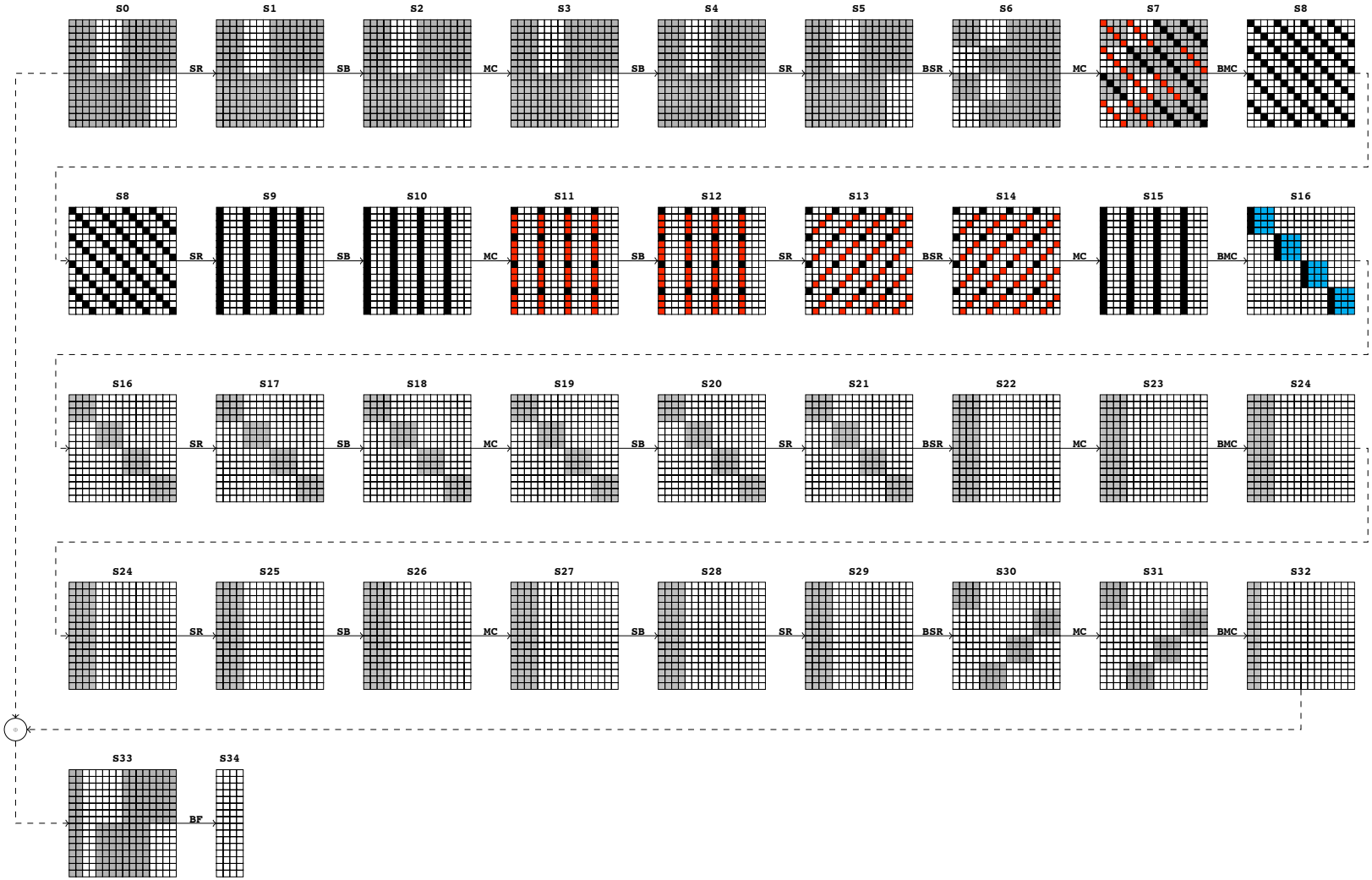
Step 2 : pair for the second subpart

Input values for one SuperSBox (out of 16)

- Input and output differences are known
- $\mathbb{P}(\textit{success}) \approx 0.5$
- Similar as [SLWSO-A10] with non-full-active input
- Avoid rebound technique on the SuperSBox
- Time : 2^{11} computations (naive : 2^{32})



Step 3 : Find remaining values for the merge to be possible

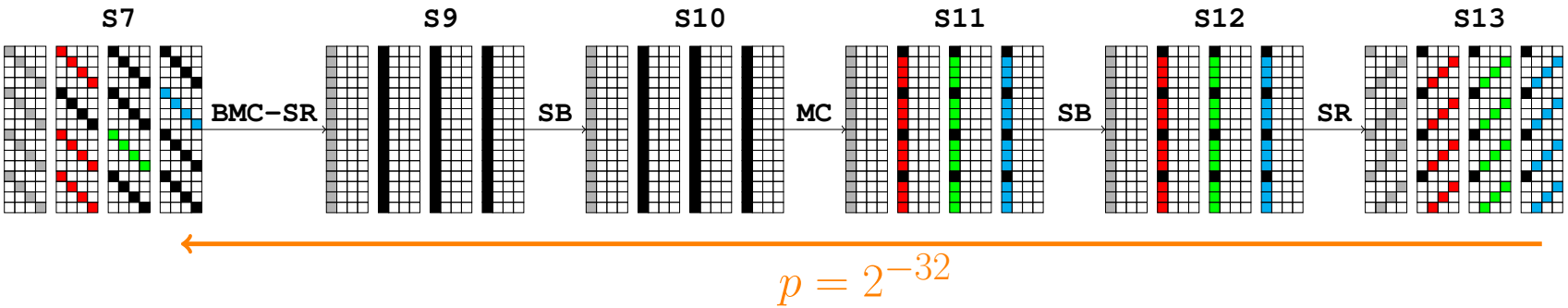


Step 3 : Find remaining values for the merge to be possible

Known : Gray bytes (already solved)

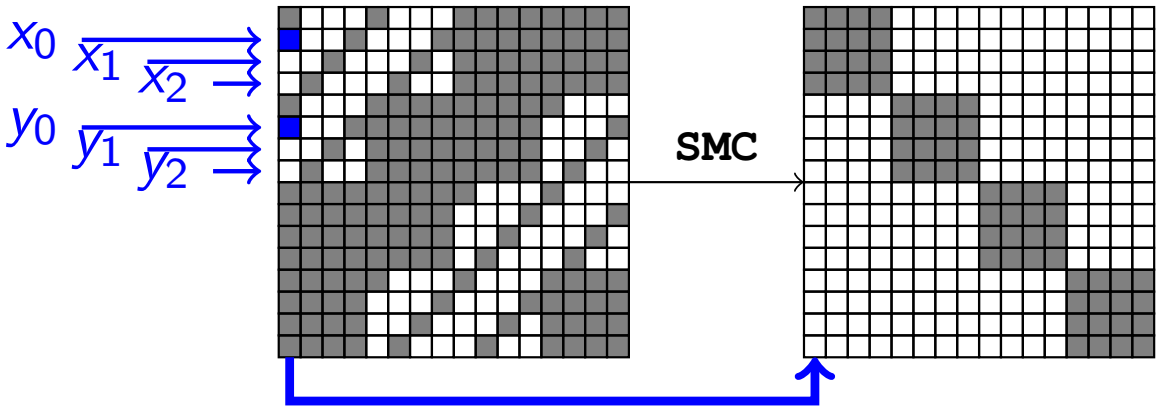
Method

1. Solve 2nd column like the first one — 2^{12}
2. Solve 3rd column (almost) like the first one — 2^{4}
3. 128-bit merging constraint $\Longrightarrow$ 4th column
4. $\mathbb{P}(\text{S9} \rightarrow \text{S7}) = \left(2^{-8}\right)^4 = 2^{-32}$ — 2^{32}
5. Try a new solution for 3rd column until success



Find all the values in the merge

Goal : Find white bytes
Known : All gray bytes



$$\begin{cases} L_0(x_0, y_0) & = c_0 \\ L_1(x_1, y_1) & = c_1 \\ L_2(x_2, y_2) & = c_2 \end{cases}$$

Solving

- Three independent 8-bit constraints for each slice
- Merge done in 2^{32} computations

Conclusion

- Attack on 4-round ECHO-256 compression function
- Based on the nice idea of the SuperMixColumns
- **Collisions** : 2^{52} computations
  - First subpart : 2^{36}
  - Second subpart : 2^{27}
  - Feed-forward : 2^{52}
- **Near-collisions** : 2^{36} computations
  - First subpart : 2^{36}
  - Second subpart : 2^{27}
  - Feed-forward : 2^{36}
- Low memory complexity : 2^{16}
- Attack implemented and validated (20k lines of C)

Conclusion

- Attack on 4-round ECHO-256 compression function
- Based on the nice idea of the SuperMixColumns
- **Collisions** : 2^{52} computations
  - First subpart : 2^{36}
  - Second subpart : 2^{27}
  - Feed-forward : 2^{52}
- **Near-collisions** : 2^{36} computations
  - First subpart : 2^{36}
  - Second subpart : 2^{27}
  - Feed-forward : 2^{36}
- Low memory complexity : 2^{16}
- Attack implemented and validated (20k lines of C)

Thank you !

Near-Collision example

$S[i,j]$	h_i	h'_i	$h_i + h'_i$
$S0[0,0]$	DEDF73AC E834ABF3 1DA654E7 8B80E057	DEDF73AC E834ABF3 1DA654E7 8B80E057	....
$S0[1,0]$	8C82AF64 E938032D EA498F65 4F3FA168	8C82AF64 E938032D EA498F65 4F3FA168	....
$S0[2,0]$	A3DEC6EE BDD97F9C 69425DE7 B88FAE55	A3DEC6EE BDD97F9C 69425DE7 B88FAE55	....
$S0[3,0]$	E0276510 531114BA 8EA8ADD3 9037426B	E0276510 531114BA 8EA8ADD3 9037426B	....

$S[i,j]$	m	m'	$m + m'$
$S0[0,1]$	B1B7D769 8B7AD57A 7B57FF05 472BECEF	B1B7D769 8B7AD57A 7B57FF05 472BECEF	....
$S0[1,1]$	D9E41EF0 FB869029 29B437B2 CC398919	D9E41EF0 FB869029 29B437B2 CC398919	....
$S0[2,1]$	CAAAC63A E8B4F522 DCA83BB4 52227A82	B6477E77 581C4385 A0035D3E 8C061217	7CEDB84D B0A8B6A7 7CAB668A DE246895
$S0[3,1]$	9142CAB0 D8421346 E35702E9 477A5AAB	6104E89C 8E995FCC 2AF9D466 B2C3D16C	F046222C 56DB4C8A C9AED68F F5B98BC7
$S0[0,2]$	F097871F B8733C73 3BD02C4C F7004240	A1E83191 315E7268 04D6F3D6 BF87220C	517FB68E 892D4E1B 3F06DF9A 4887604C
$S0[1,2]$	A765E039 EB6C558F B444631F DD4BC1AB	6993F70F 5F87B6BF 6402FB87 CA7859C6	CEF61736 B4EBE330 D0469898 1733986D
$S0[2,2]$	BCEAEFAA 8304B57E F2C6732D D396D8F8	2507A8FD 67F83C71 9B523FBF 3534F32E	99ED4757 E4FC890F 69944C92 E6A22BD6
$S0[3,2]$	C406CB83 EA157529 E008A7CB 11675D1A	005DF381 40322440 16E70F34 454F1318	C45B3802 AA275169 F6EFA8FF 54284E02
$S0[0,3]$	84258159 7A87E98E B750B21D 31D0F510	0429D2E3 5B02D7DE A22839AA 174013DA	800C53BA 21853E50 15788BB7 2690E6CA
$S0[1,3]$	A5808F25 DBDE4281 ECAFEF87 3607ACBB	8EEC6709 3B61D819 29D65D83 09B27795	2B6CE82C E0BF9A98 C579B204 3FB5DB2E
$S0[2,3]$	E9B4133F F7C776FC E9F2C741 754EBC6B	E9B4133F F7C776FC E9F2C741 754EBC6B	....
$S0[3,3]$	8C219844 7E17C475 7AED625F 3B685665	8C219844 7E17C475 7AED625F 3B685665	....

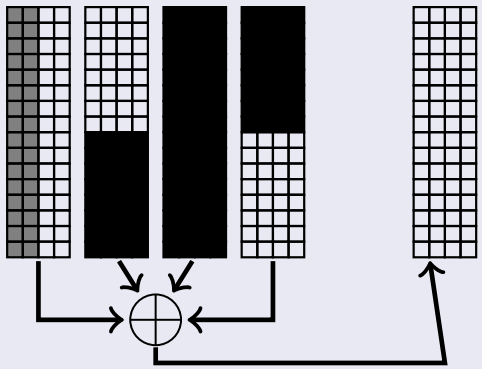
$S[i,j]$	h_{i+1}	h'_{i+1}	$h_{i+1} + h'_{i+1}$
$S34[0,0]$	0EC3168C C7F787CA 4006FA09 3E29BA5E	0E55168C C7F714CA 4006FA0E C129BA5E	..96.... ....93.. ......07 FF......
$S34[1,0]$	FF729D65 2B555D10 AD0CF15C 9A9AFF87	FF179D65 2B55D810 AD0CF1D5 779AFF87	..65.... ....85.. ......89 ED......
$S34[2,0]$	7E2C1C9D 542E3BE0 AF880377 8887502A	7ED31C9D 542EF8E0 AF88037A 7587502A	..FF.... ....C3.. ......0D FD......
$S34[3,0]$	A776FCAF 96C2F792 FF051583 FF6482C6	A771FCAF 96C2F592 FF0515CC 0A6482C6	..07.... ....02.. ......4F F5......

Feed-Forward

Compression step $\Longrightarrow$ Collision

- Some differences are known
- Constraints on differences of the 1st round
- ECHO-rows are independent
- Freedom at the input of the SuperSBox

BigFinal



Finding input values for SuperSBoxes such that :

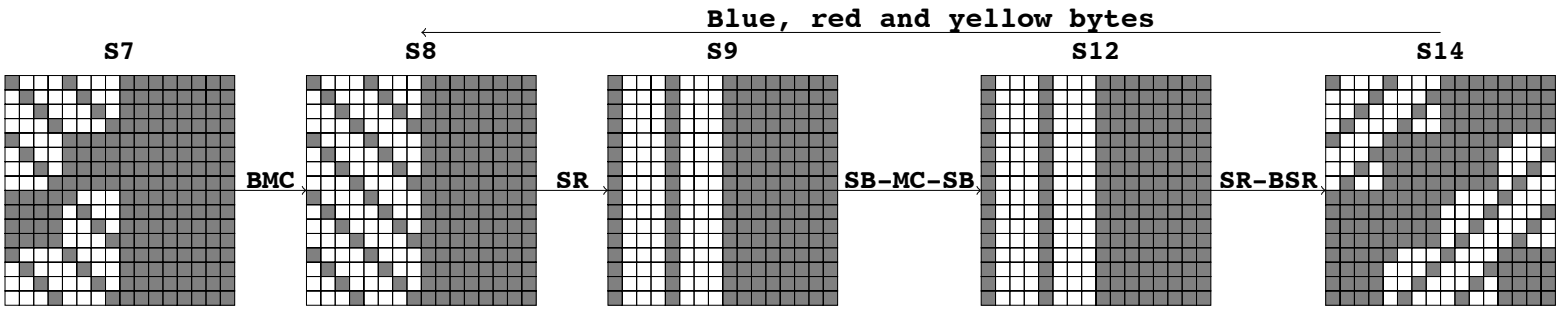
- All differences cancel out in the feed-forward 2^{32} per row
- Time : 2^{52}

Find all the values in the merge

Goal : Find white bytes with constraints
Known : All gray bytes

Method

1. Randomize blue, red and yellow bytes in S14
2. Propagate them backwards through SuperSBox
3. Deduce green bytes in S8
4. Progagate them forwards
5. Check 32-bit condition. Go to 1. 2^{32}

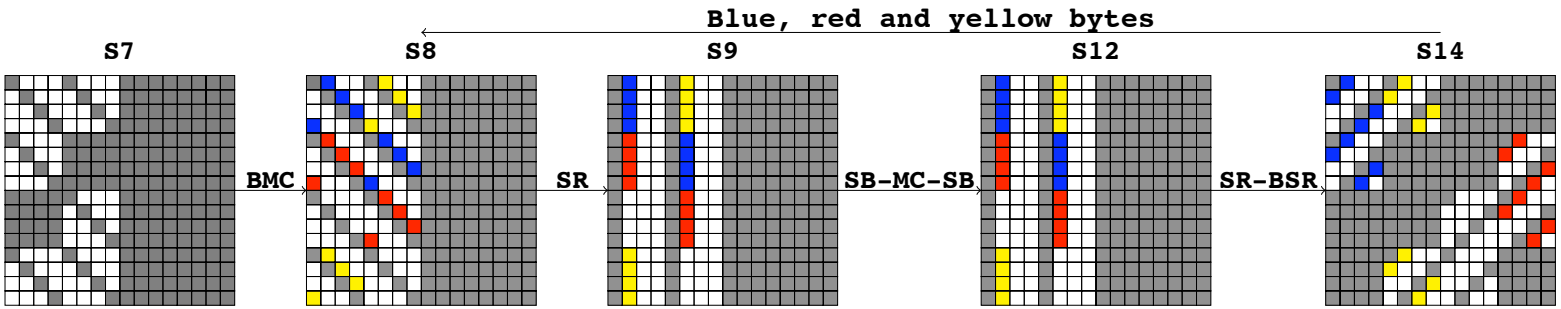


Find all the values in the merge

Goal : Find white bytes with constraints
Known : All gray bytes

Method

1. Randomize blue, red and yellow bytes in S14
2. Propagate them backwards through SuperSBox
3. Deduce green bytes in S8
4. Progagate them forwards
5. Check 32-bit condition. Go to 1. 2^{32}

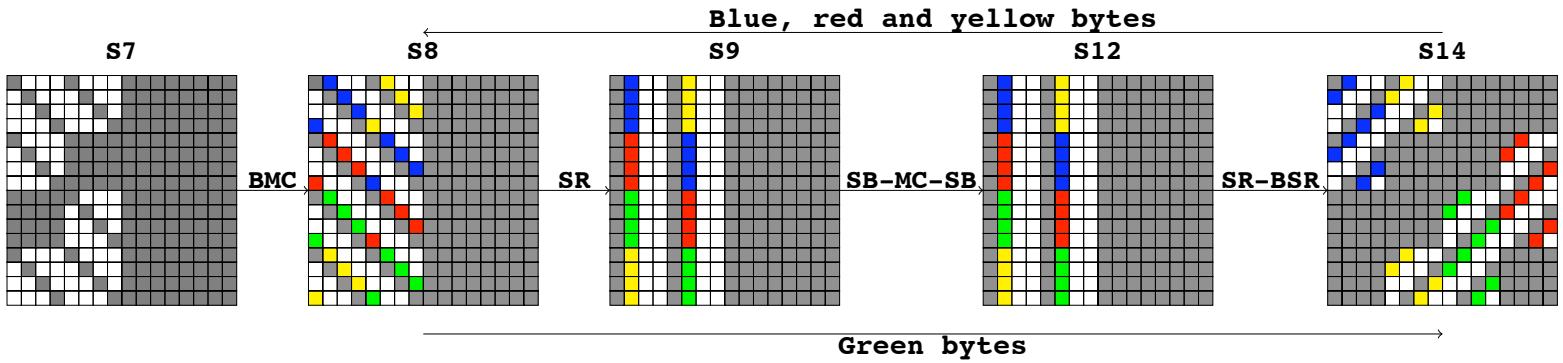


Find all the values in the merge

Goal : Find white bytes with constraints
Known : All gray bytes

Method

1. Randomize blue, red and yellow bytes in S14
2. Propagate them backwards through SuperSBox
3. Deduce green bytes in S8
4. Progagate them forwards
5. Check 32-bit condition. Go to 1. 2^{32}

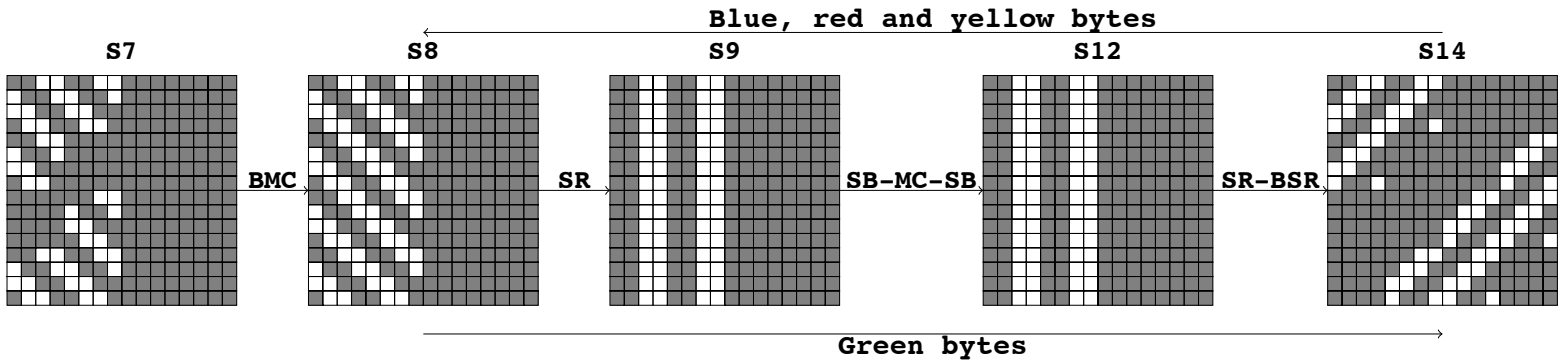


Find all the values in the merge

Goal : Find white bytes with constraints
Known : All gray bytes

Method

1. Randomize blue, red and yellow bytes in S14
2. Propagate them backwards through SuperSBox
3. Deduce green bytes in S8
4. Progagate them forwards
5. Check 32-bit condition. Go to 1. 2^{32}

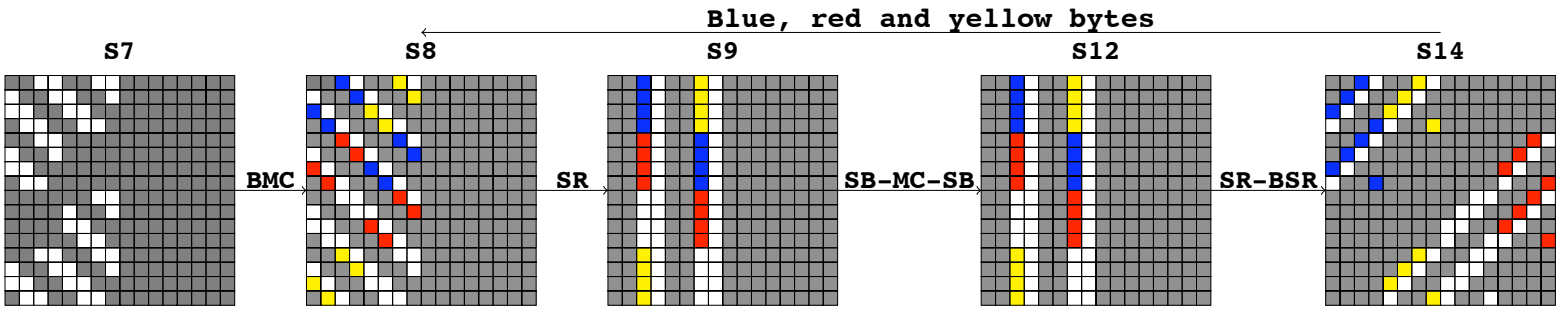


Find all the values in the merge

Goal : Find white bytes with constraints
Known : All gray bytes

Method

1. Randomize blue, red and yellow bytes in S14
2. Propagate them backwards through SuperSBox
3. Deduce green bytes in S8
4. Progagate them forwards
5. Check 32-bit condition. Go to 1. 2^{32}

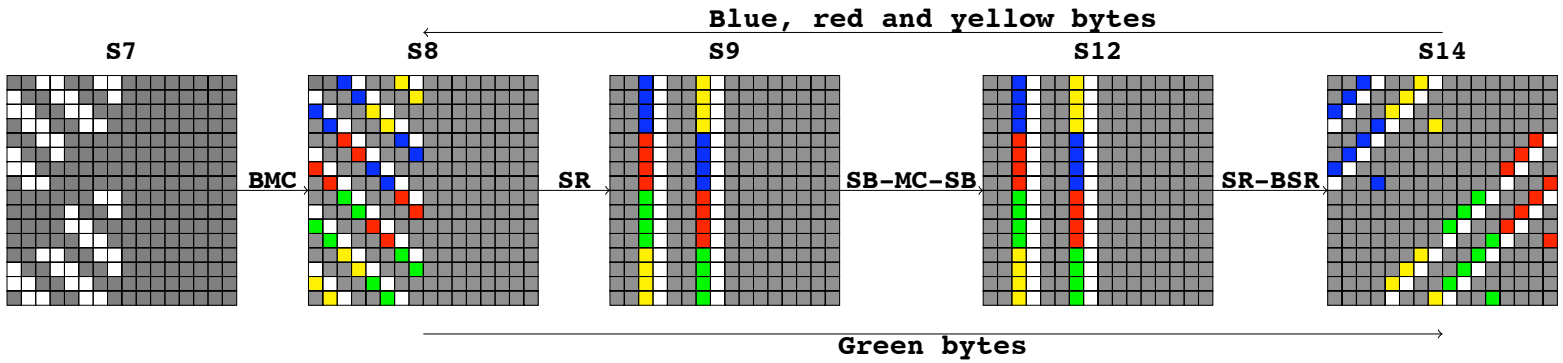


Find all the values in the merge

Goal : Find white bytes with constraints
Known : All gray bytes

Method

1. Randomize blue, red and yellow bytes in S14
2. Propagate them backwards through SuperSBox
3. Deduce green bytes in S8
4. Progagate them forwards
5. Check 32-bit condition. Go to 1. 2^{32}

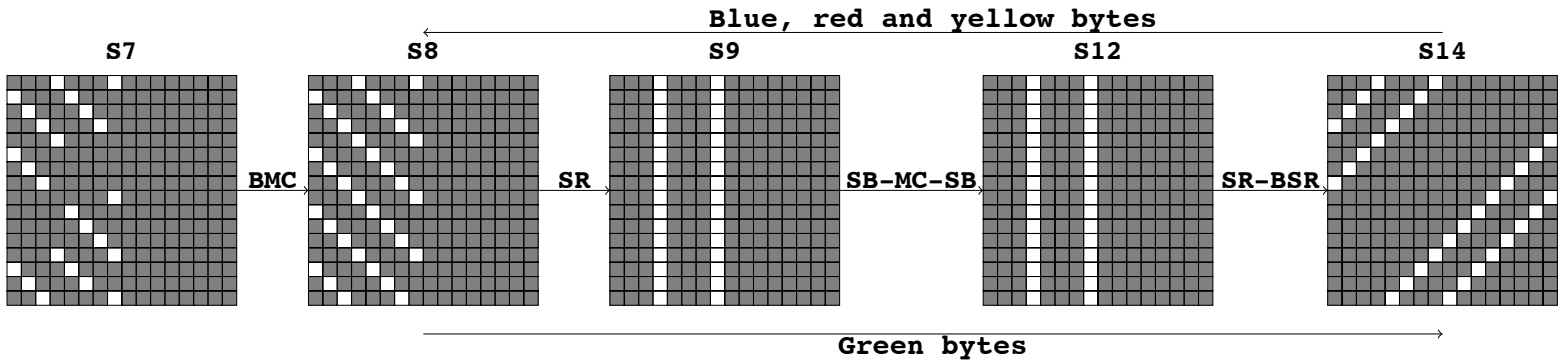


Find all the values in the merge

Goal : Find white bytes with constraints
Known : All gray bytes

Method

1. Randomize blue, red and yellow bytes in S14
2. Propagate them backwards through SuperSBox
3. Deduce green bytes in S8
4. Progagate them forwards
5. Check 32-bit condition. Go to 1. 2^{32}

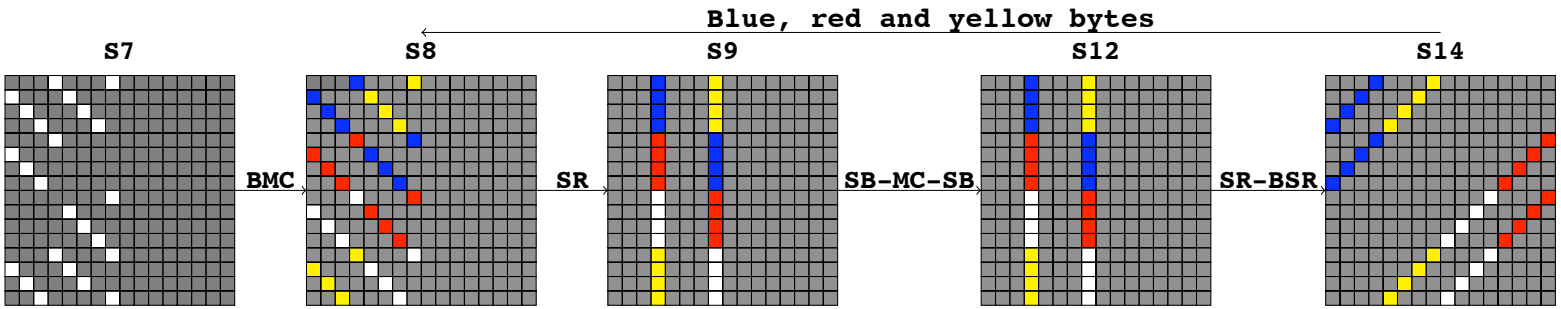


Find all the values in the merge

Goal : Find white bytes with constraints
Known : All gray bytes

Method

1. Randomize blue, red and yellow bytes in S14
2. Propagate them backwards through SuperSBox
3. Deduce green bytes in S8
4. Progagate them forwards
5. Check 32-bit condition. Go to 1. 2^{32}

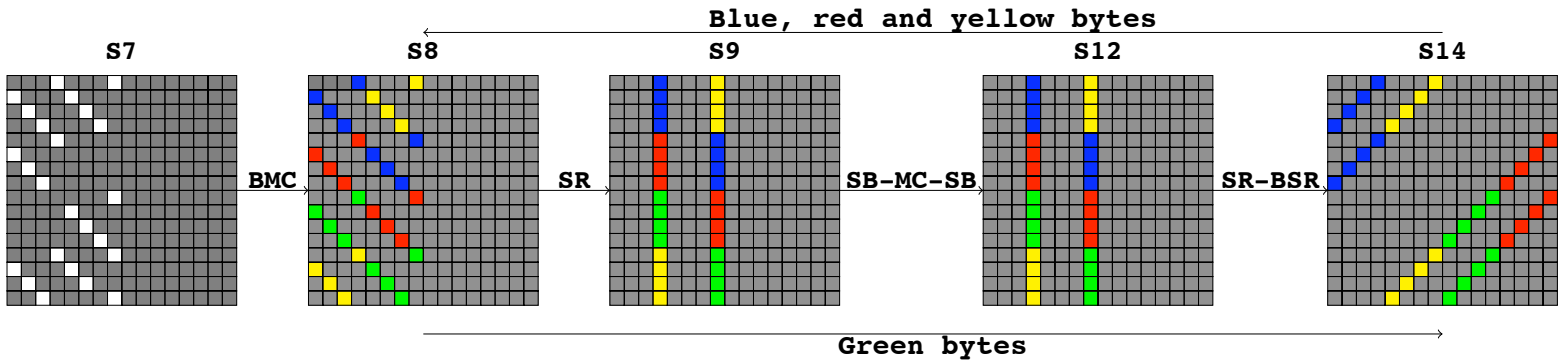


Find all the values in the merge

Goal : Find white bytes with constraints
Known : All gray bytes

Method

1. Randomize blue, red and yellow bytes in S14
2. Propagate them backwards through SuperSBox
3. Deduce green bytes in S8
4. Progagate them forwards
5. Check 32-bit condition. Go to 1. 2^{32}



Find all the values in the merge

Goal : Find white bytes with constraints
Known : All gray bytes

Method

1. Randomize blue, red and yellow bytes in S14
2. Propagate them backwards through SuperSBox
3. Deduce green bytes in S8
4. Progagate them forwards
5. Check 32-bit condition. Go to 1. 2^{32}