Extending AES Improvements: A Proposal for Alpha-MAC in View of Collision Resistance

Adrián Alfonso Peñate and Pablo Freyre Arrozarena
Institute of Cryptography, University of Havana, Cuba

9th Workshop on Current Trends in Cryptology
September 15-17, 2020

Introduction I - What are Message Authentication Codes?

Message Authentication Codes, or MAC functions, are symmetric primitives used to ensure authenticity of messages.

PROCEDURE

Sender: Computes $Tag = MAC(Message, Key)$ and sends Tag.

Receiver: Computes $Out = MAC(Message, Key)$ and checks that

- $Out = Tag$ the message came from the stated sender.
- $Out \neq Tag$ the message did not come from the stated sender or it has been changed.

INTERNAL COLLISION

Sender: Computes $Tag = MAC(Message, Key)$ and sends Tag.

Attacker: Finds a $Message$ such that $Tag = MAC(Message, Key)$.

Receiver: Computes $Out = MAC(Message, Key)$ and checks that $Out = Tag$ but the message did not come from the sender.

Introduction II - What our Paper is About

In 2005 it is presented a kind of construction for MAC functions, named Alred, as well as the specific instance Alpha-MAC with the standard AES as underlying primitive.

Daemen, J. and Rijmen, V. A New MAC Construction Alred and a Specific Instance Alpha-MAC. LNCS 3557, pp. 1–17, 2005.

In 2006 it is presented an attack on Alpha-MAC exploiting its algebraic properties to find internal collisions.

Huang, J., Seberry, J. and Susilo, W. On the Internal Structure of Alpha-MAC. LNCS 4341, pp. 271–285, 2006.

In this paper we compute the success probability of the attack with a change in the internal structure of Alpha-MAC.

Internal Structure of Alpha-MAC

Input: message $x = x_1 x_2 \cdots x_q$ and secret key k
begin
  $z_0 = \mathrm{AES}_k(0)$
  for i **from** 1 **to** q **do**
    $z_i = \mathrm{Round}(z_{i-1}, \mathrm{Injection}(x_i))$
  end for
  $\mathrm{tag} = \mathrm{Truncation}(\mathrm{AES}_k(z_q))$
end
Output: message tag Alpha-MAC$(x, k) = \mathrm{tag}$

In the general construction, Alred, any block cipher can be used instead of the standard AES.

Internal Structure of Alpha-MAC

The unkeyed round function of Alpha-MAC acts on each intermediate state like the round function of the standard AES

$$z_i = \text{AddRoundKey}(\text{MixColumns}(\text{ShiftRows}(\text{SubBytes}(z_{i-1}))), \text{Injection}(x_i))$$

through the following transformations:

(1) SubBytes: The S-box on each byte.

(2) ShiftRows: The cyclic rotation of each row.

(3) MixColumns: The MDS matrix on each column.

(4) AddRoundKey: The bitwise XOR with the message injection, where

$$\text{Injection}(x_i) = (x_i^1, 0, x_i^2, 0, 0, 0, 0, 0, x_i^3, 0, x_i^4, 0, 0, 0, 0, 0)$$

for all 32-bit message block $x_i = (x_i^1, x_i^2, x_i^3, x_i^4)$.

For every 128 bits of message Alpha-MAC processes 4 rounds of AES, reducing in 2.5 the runtime with respect to constructions like CBC-MAC.

The Attack - Goal

It pretends to find a second preimage under the assumption that the key or an intermediate state is known, using only 5 message blocks.

Input: five message blocks $(x_{y-3}, x_{y-2}, x_{y-1}, x_y, x_{y+1})$ and the intermediate state at the input of the round $y-3$

begin

generate a second preimage $(\overline{x}_{y-3}, \overline{x}_{y-2}, \overline{x}_{y-1}, \overline{x}_y, \overline{x}_{y+1})$ randomly, guaranteeing that $\overline{x}_{y-3} \neq x_{y-3}$

perform the Backwards-aNd-Backwards search (BNB) to generate a 32-bit collision, modifying the message block $\overline{x}_{y-2}$

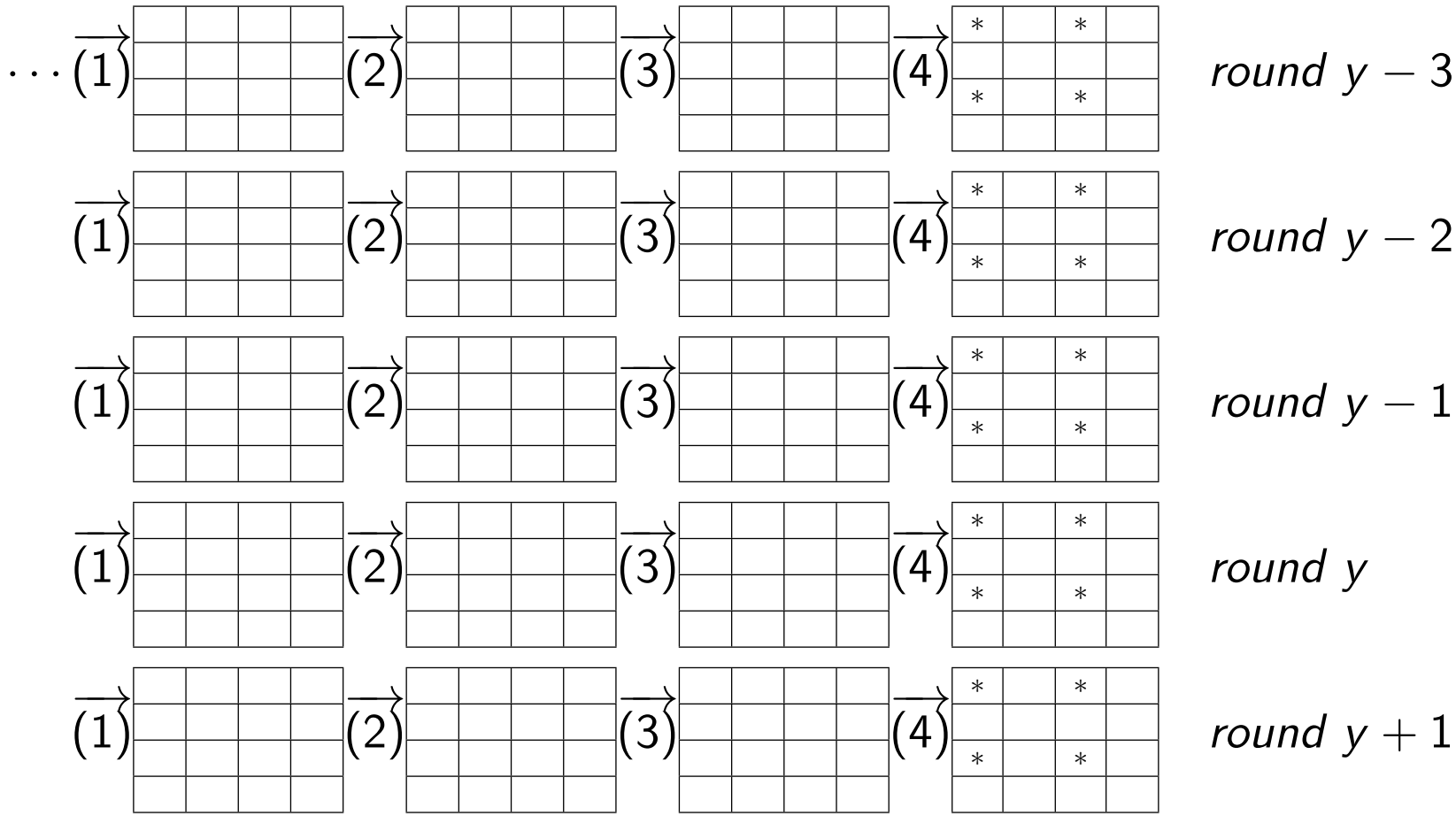
perform the Backwards-aNd-Forwards search (BNF) to extend the 32-bit collision to a 128-bit collision, modifying the message blocks $\overline{x}_{y-1}$, $\overline{x}_y$ and $\overline{x}_{y+1}$

end

Output: second preimage $(\overline{x}_{y-3}, \overline{x}_{y-2}, \overline{x}_{y-1}, \overline{x}_y, \overline{x}_{y+1})$

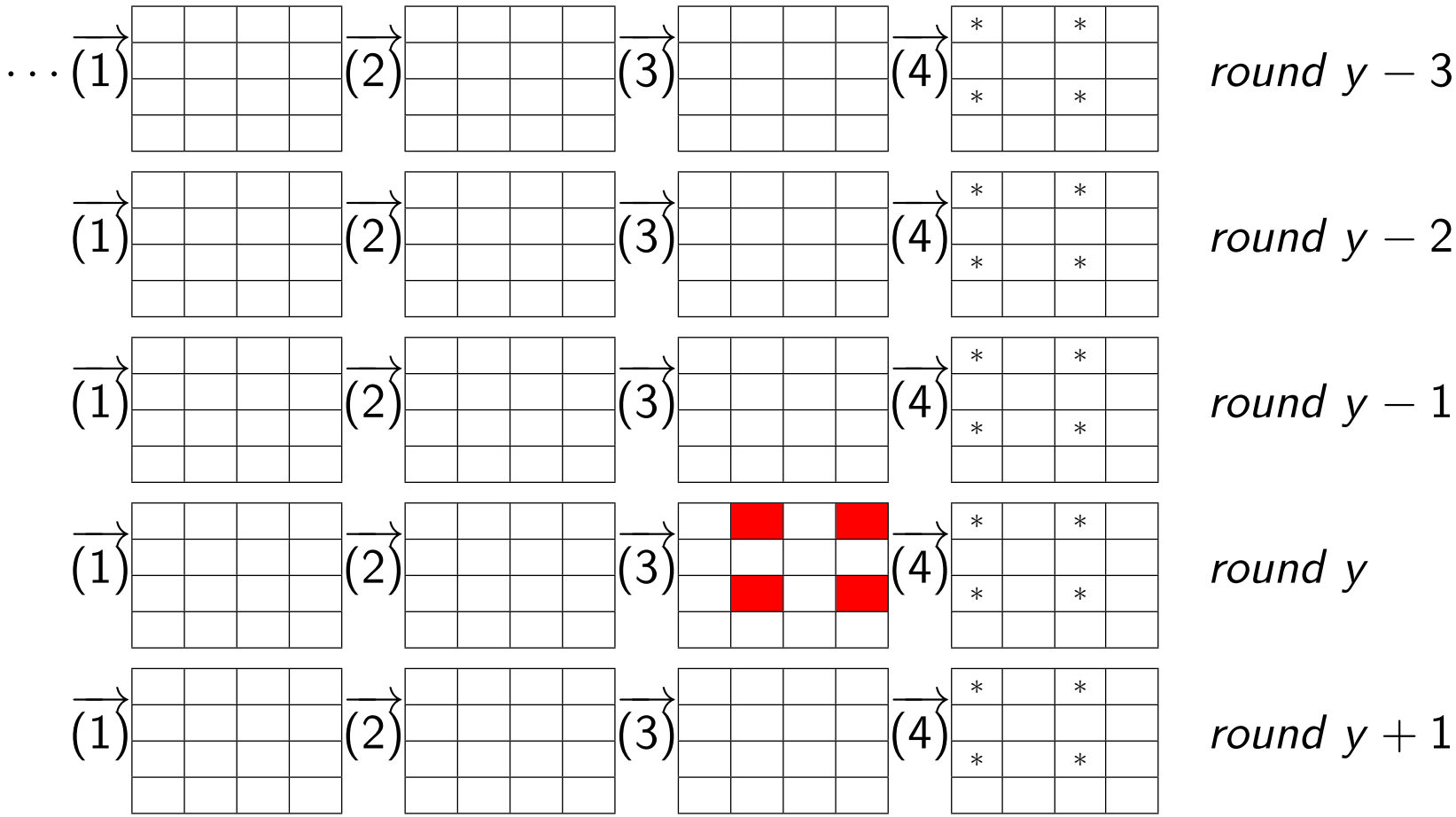
Backwards-aNd-Backwards

The BNB search finds a 32-bit collision modifying the message block $\overline{x}_{y-2}$ of the second preimage randomly generated.



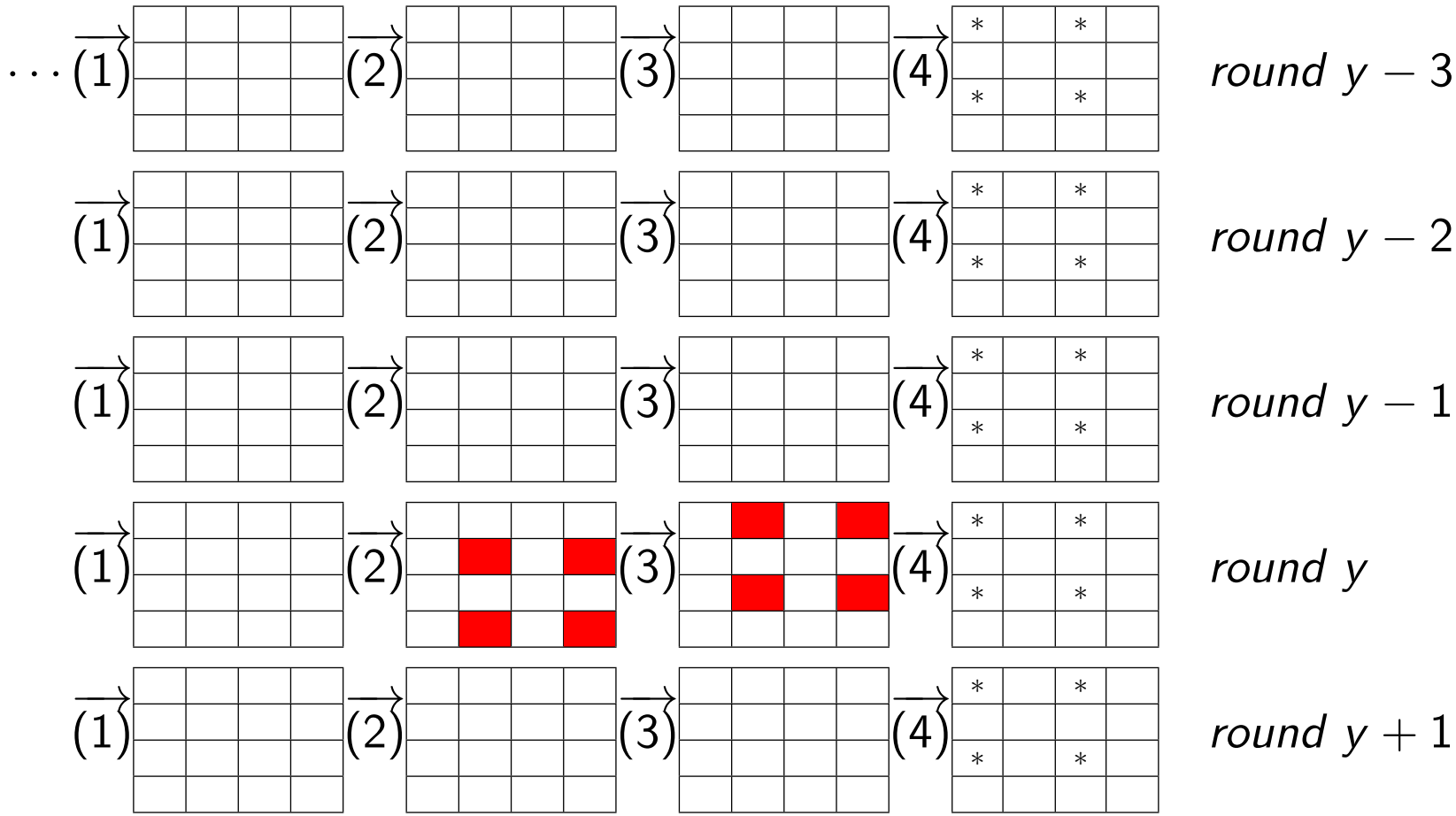
Backwards-aNd-Backwards - Step 1

It is assumed that the bytes of the positions [0,1], [0,3], [2,1] and [2,3] after MixColumns in round y collide.



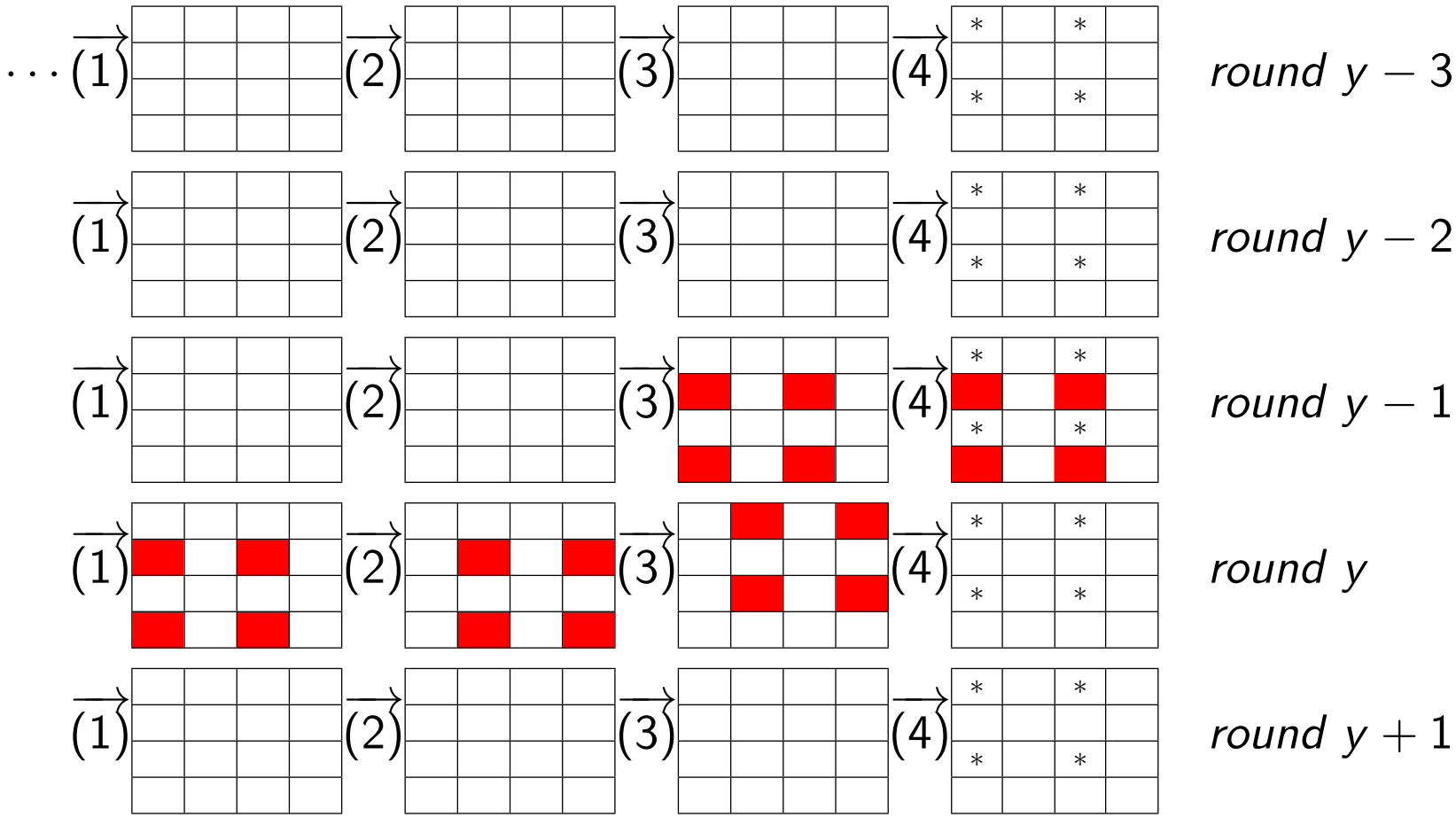
Backwards-aNd-Backwards - Step 1

Two equation systems around MixColumns in round y are solved, finding 4 collision-dependent bytes before MixColumns.



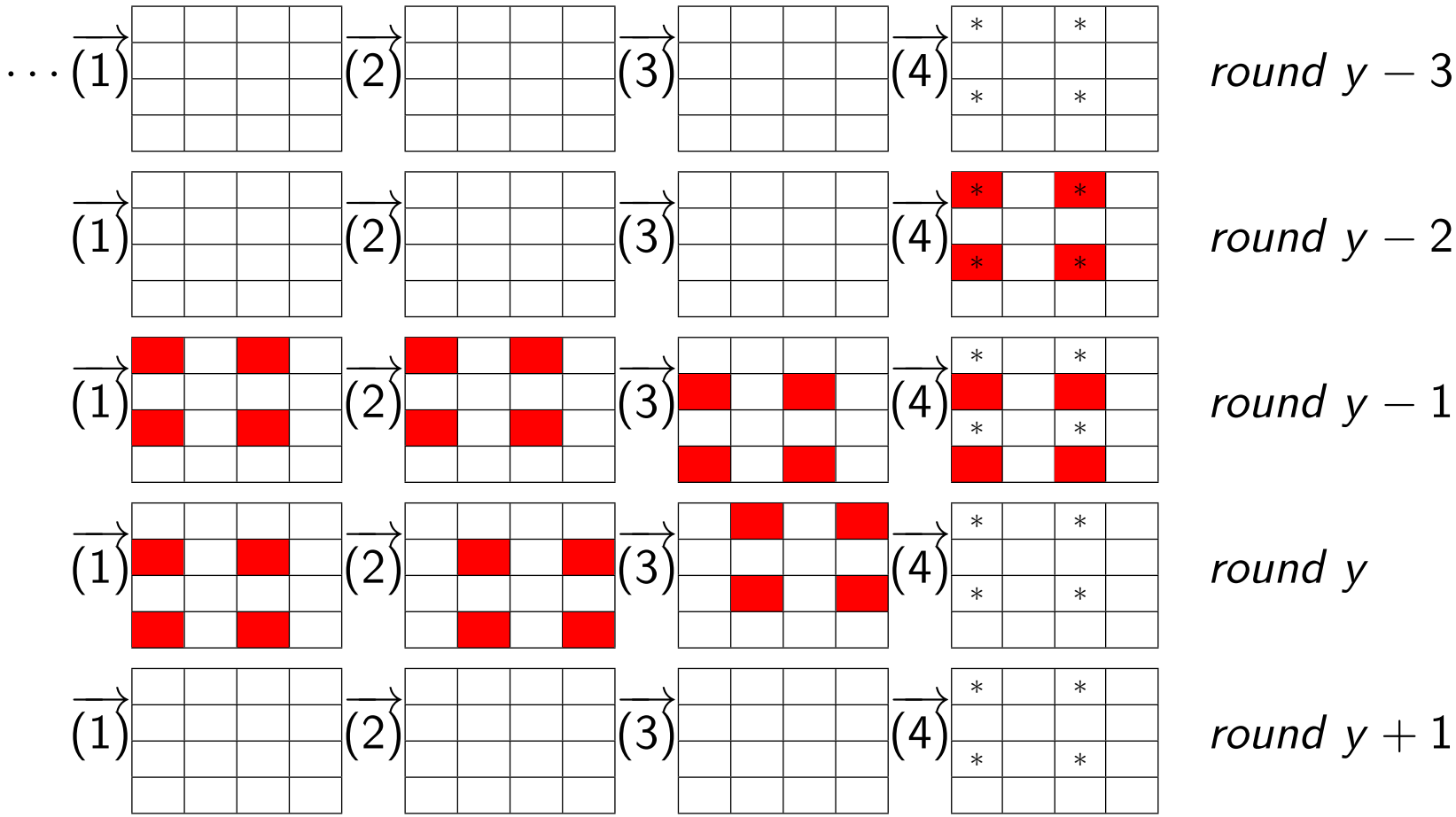
Backwards-aNd-Backwards - Step 1

The inverse transformations up to MixColumns in round $y-1$ are performed. It keeps the collision-dependency of some bytes.



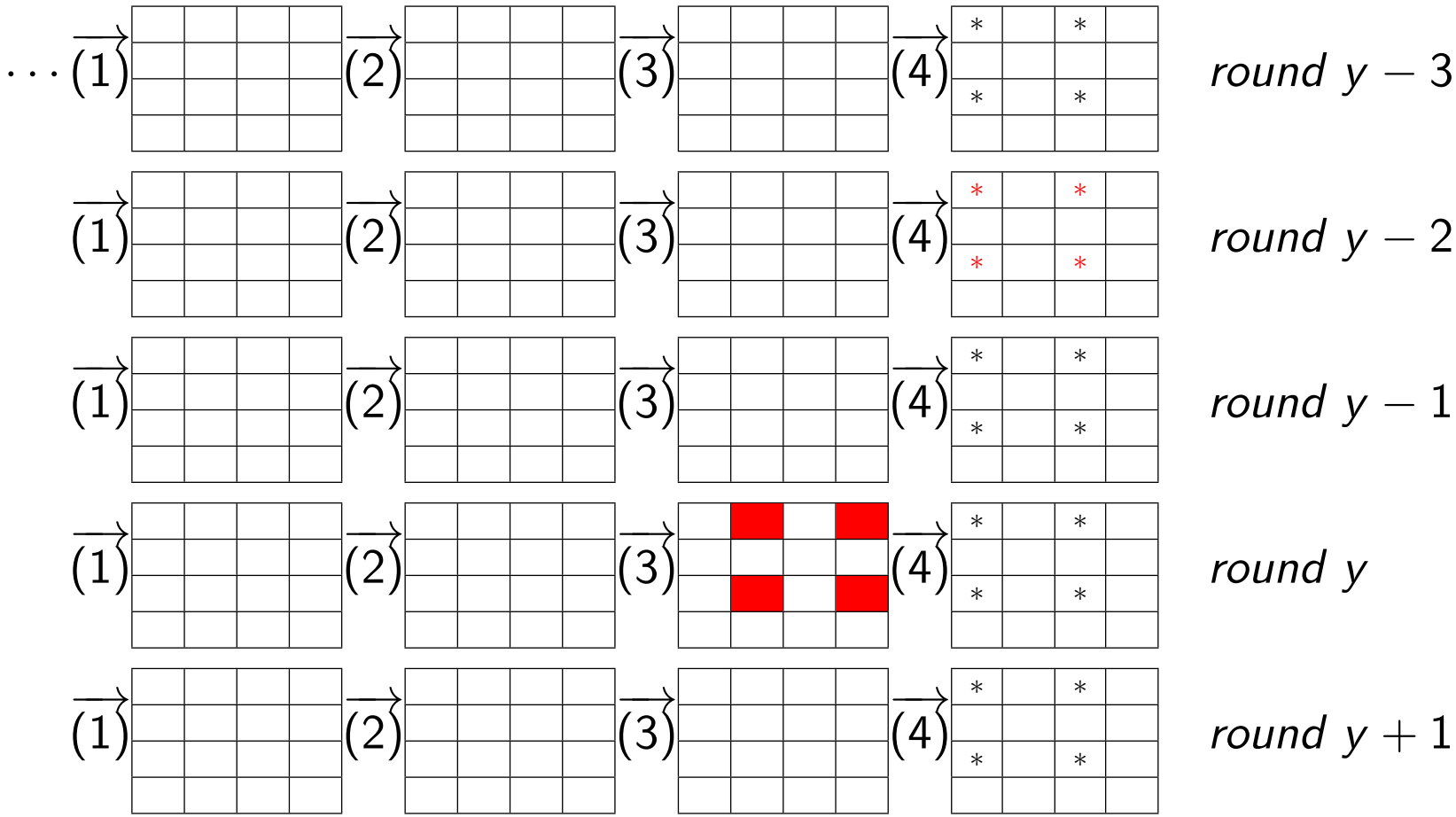
Backwards-aNd-Backwards - Step 2

Other two equation systems in the round $y-1$ are solved and the inverse transformations up to AddRoundKey in round $y-2$ are performed.



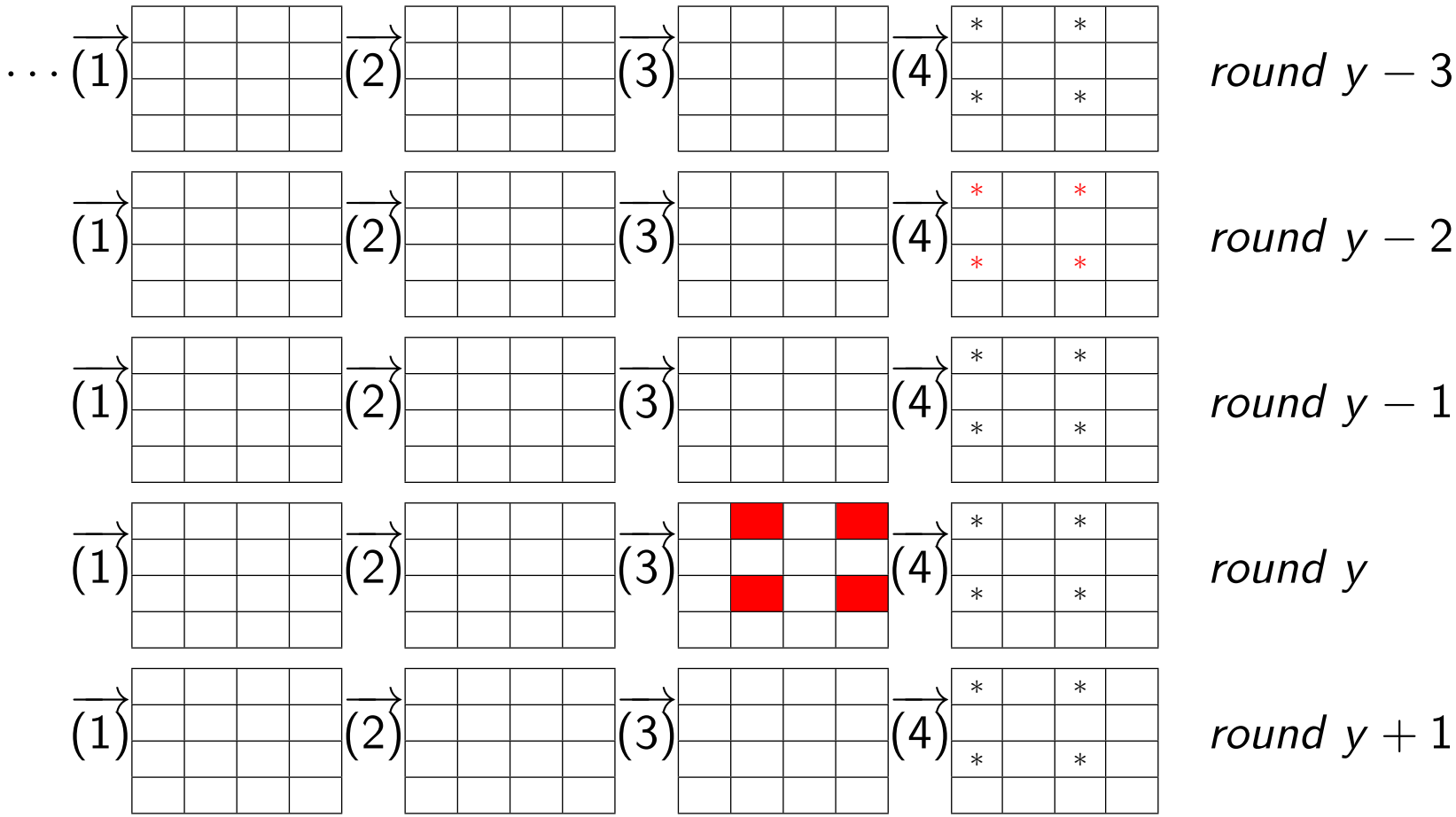
Backwards-aNd-Backwards - Step 2

The message block $\overline{x}_{y-2}$ is replaced by a collision-dependent one. At this point, both x and $\overline{x}$ collide on 32 bits after MixColumns in round y.



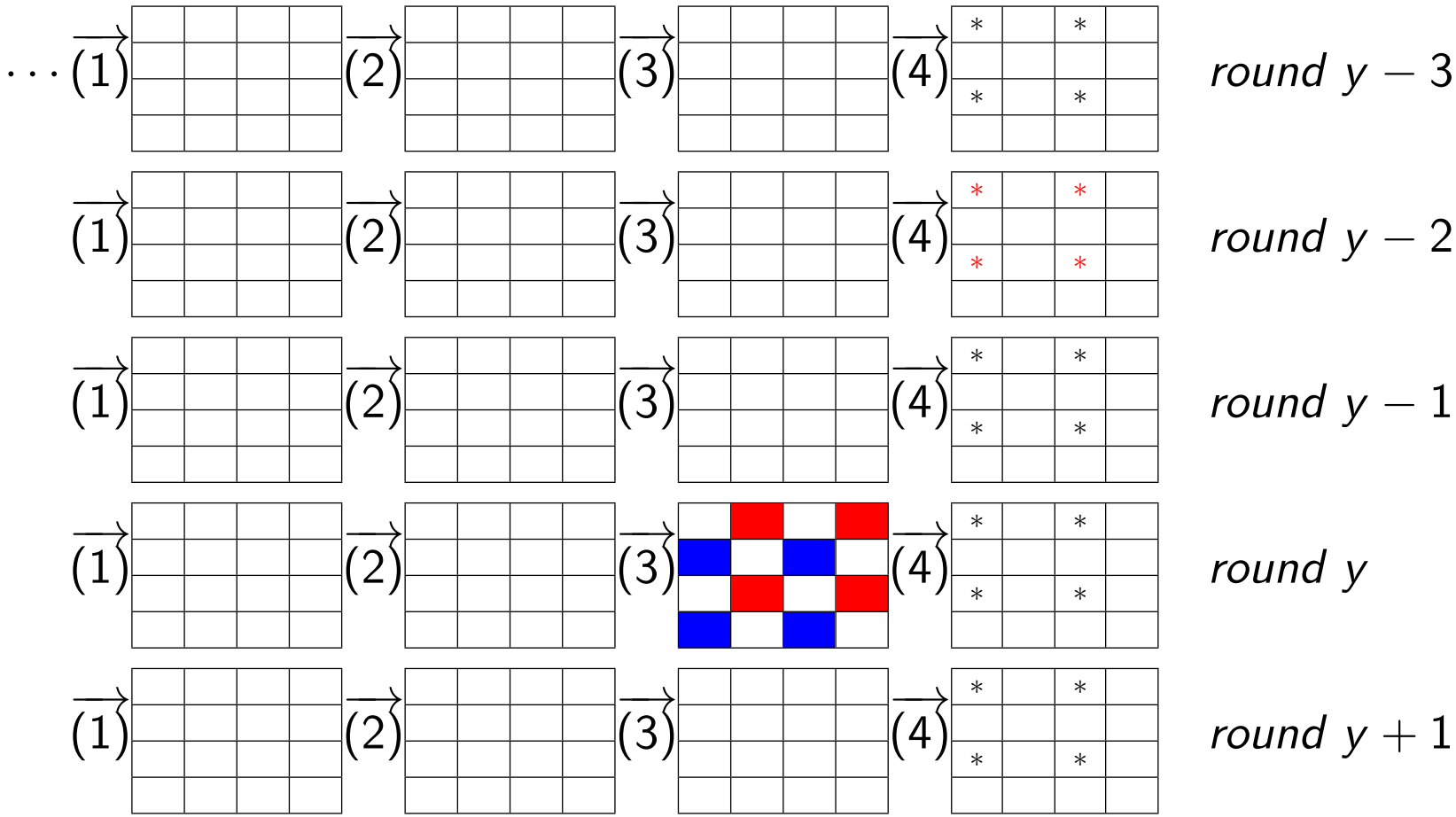
Backwards-aNd-Forwards

The BNF search extends the 32-bit collision found in the BNB search to a 128-bit collision, modifying the message blocks $\overline{x}_{y-1}$, $\overline{x}_y$ and $\overline{x}_{y+1}$.



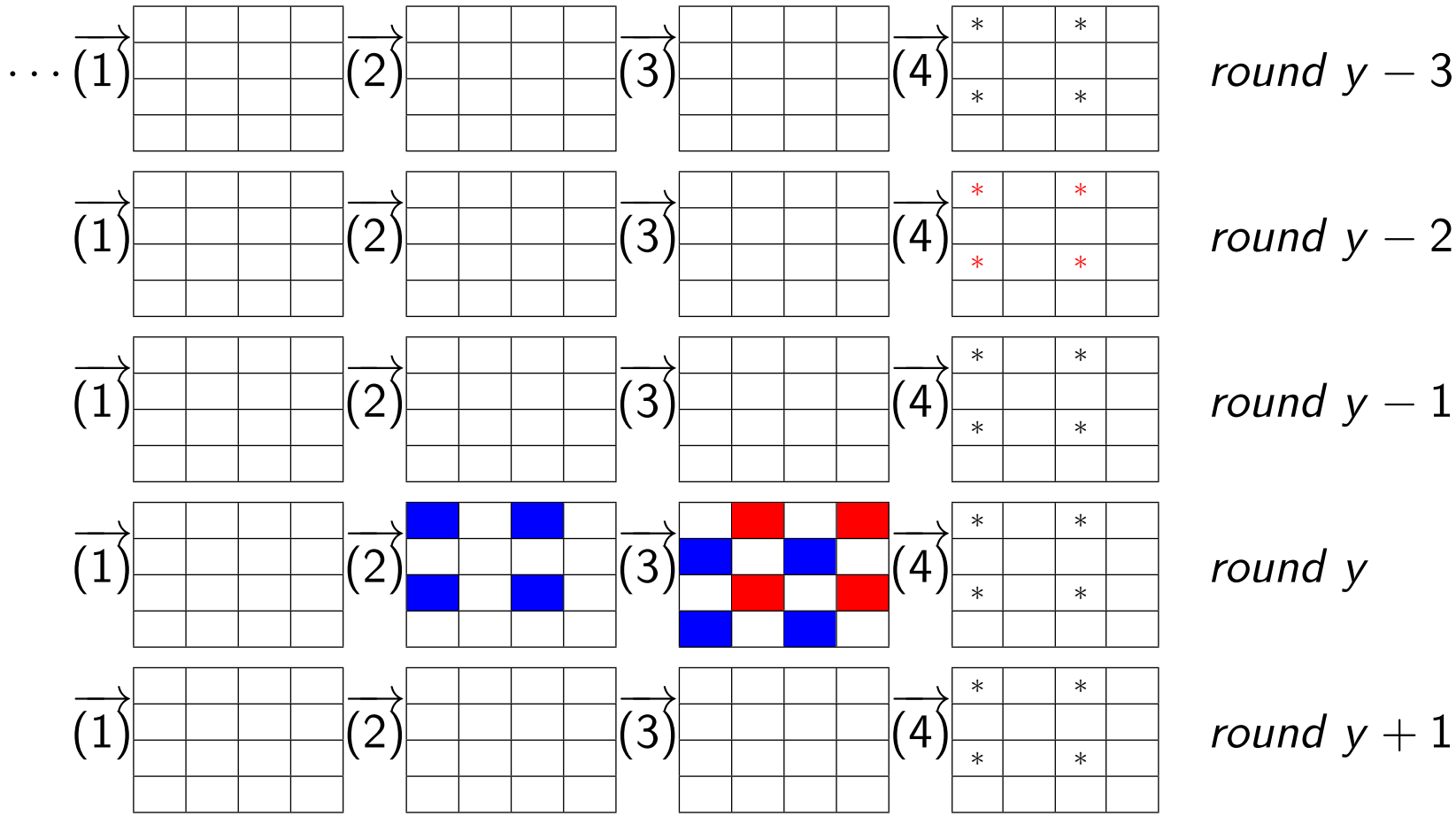
Backwards-aNd-Forwards - Step 1

It is assumed that the bytes of the positions [1,0], [1,2], [3,0] and [3,2] after MixColumns in round y collide.



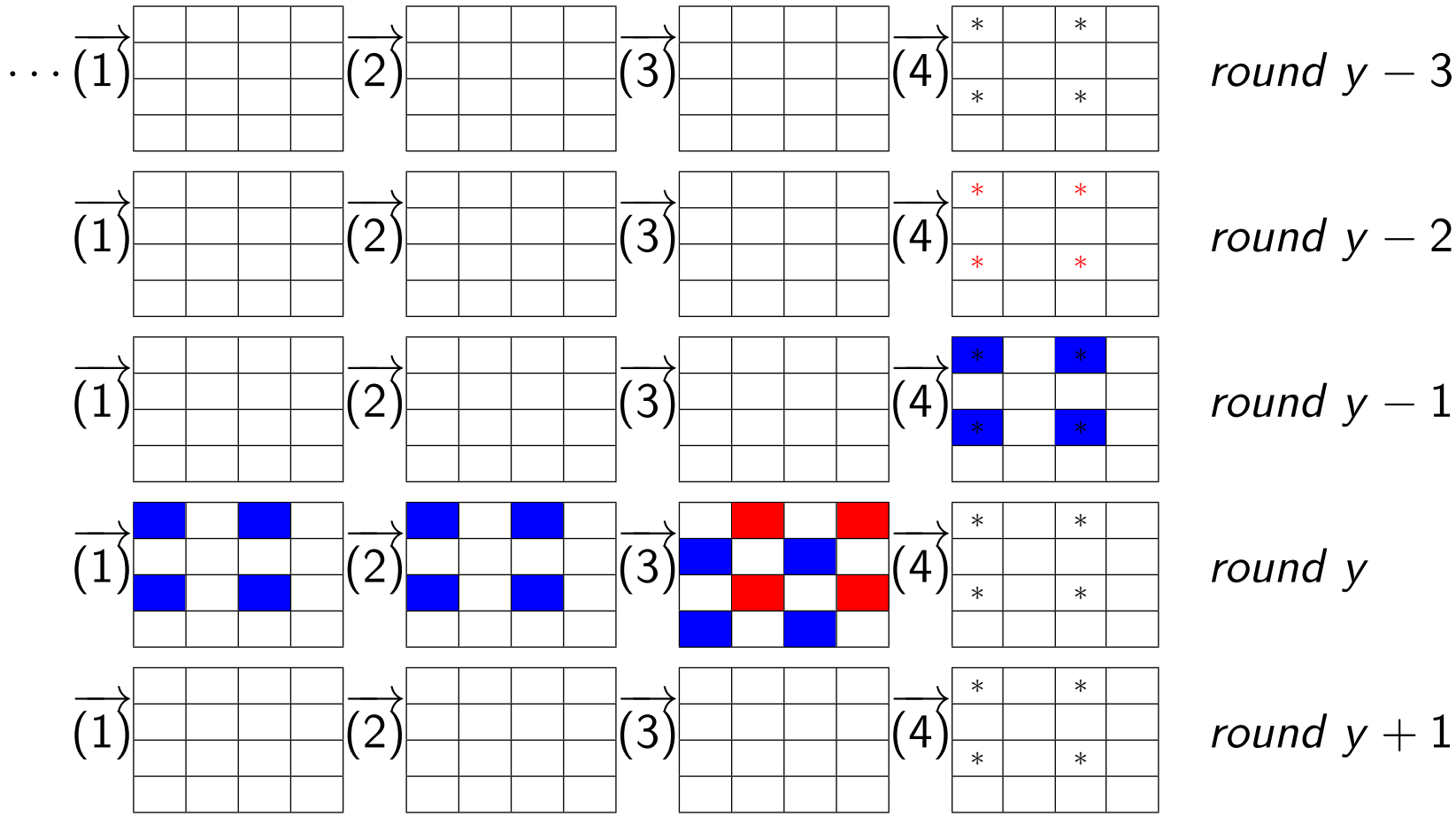
Backwards-aNd-Forwards - Step 1

Two equation systems around MixColumns in round y are solved, finding 4 collision-dependent bytes before MixColumns.



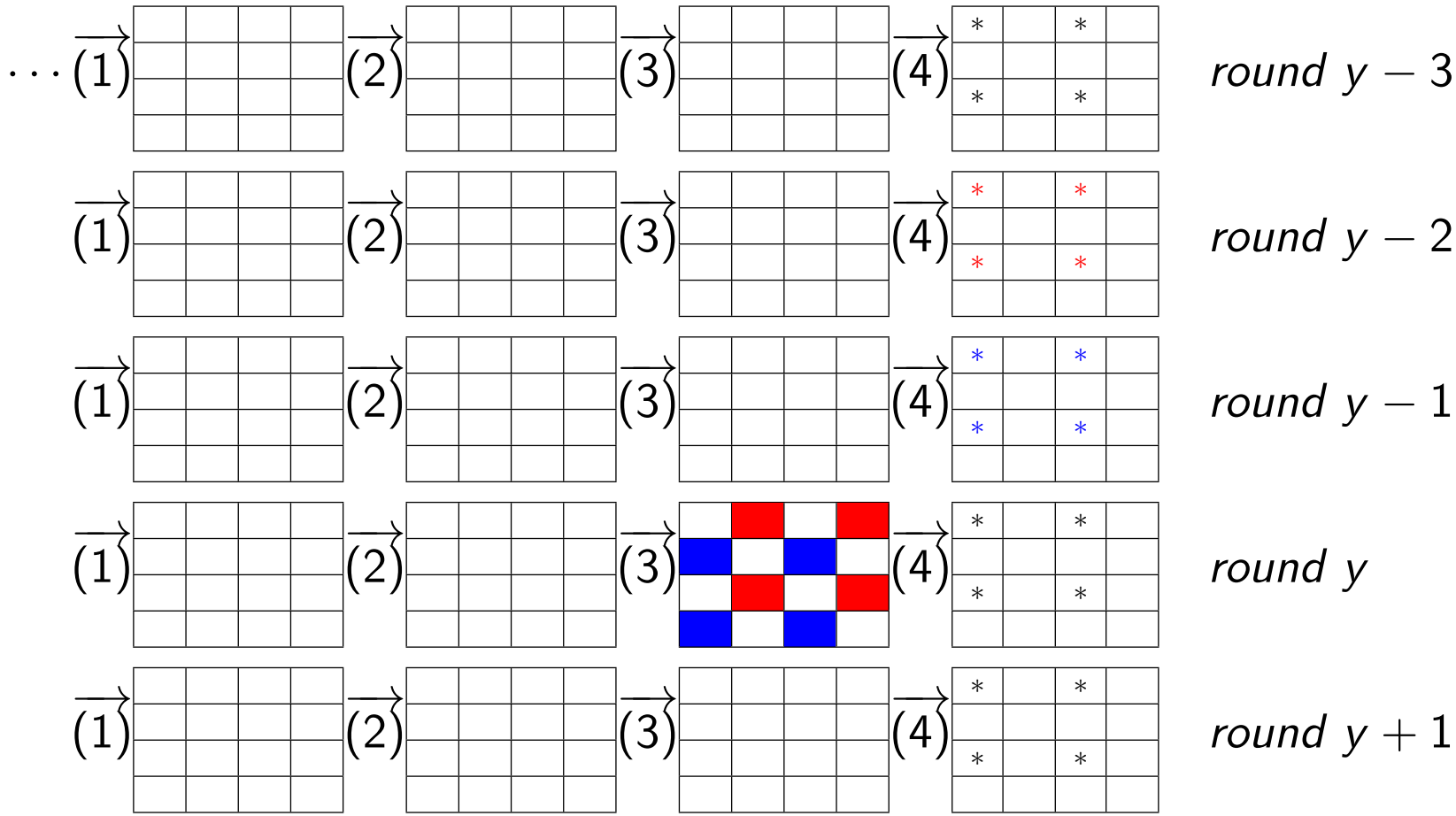
Backwards-aNd-Forwards - Step 1

The inverse transformations up to AddRoundKey in round $y-1$ are performed. It keeps the collision-dependency of some bytes.



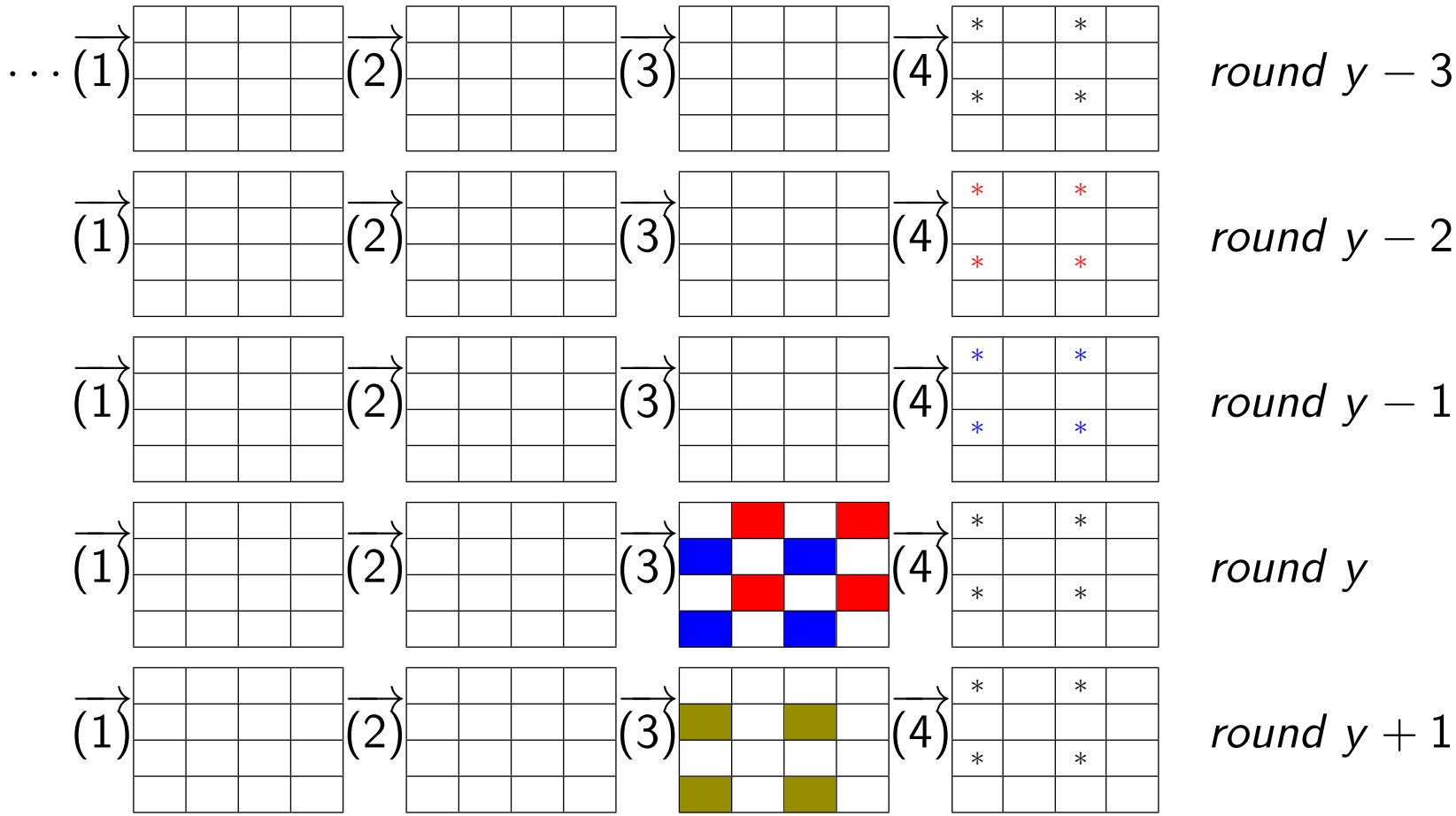
Backwards-aNd-Forwards - Step 1

The message block $\overline{x}_{y-1}$ is replaced by a collision-dependent one. At this point, both x and $\overline{x}$ collide on 64 bits after MixColumns in round y.



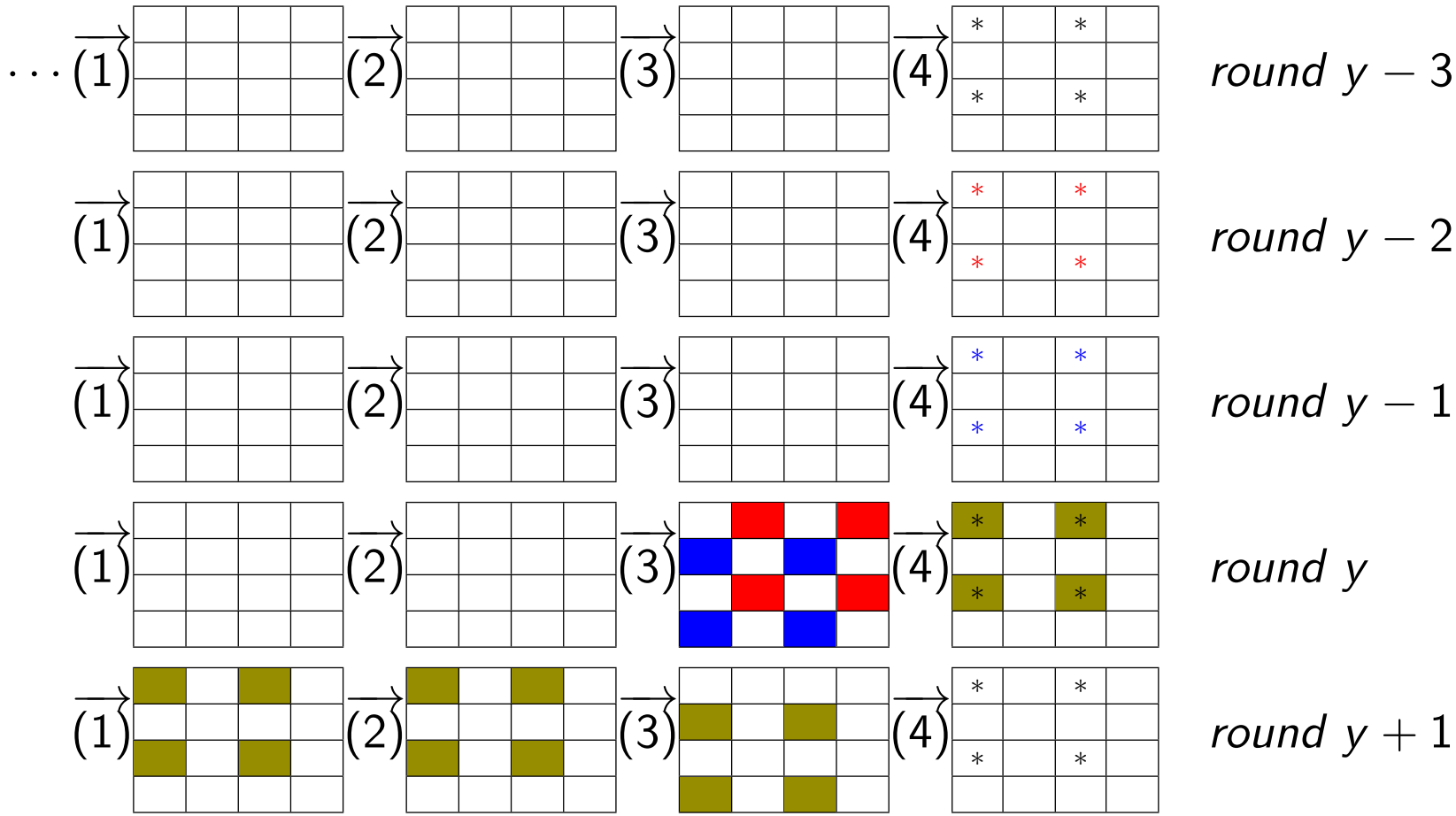
Backwards-aNd-Forwards - Step 2

Now it is assumed that the bytes of the positions [1,0], [1,2], [3,0] and [3,2] after MixColumns in round $y+1$ collide.



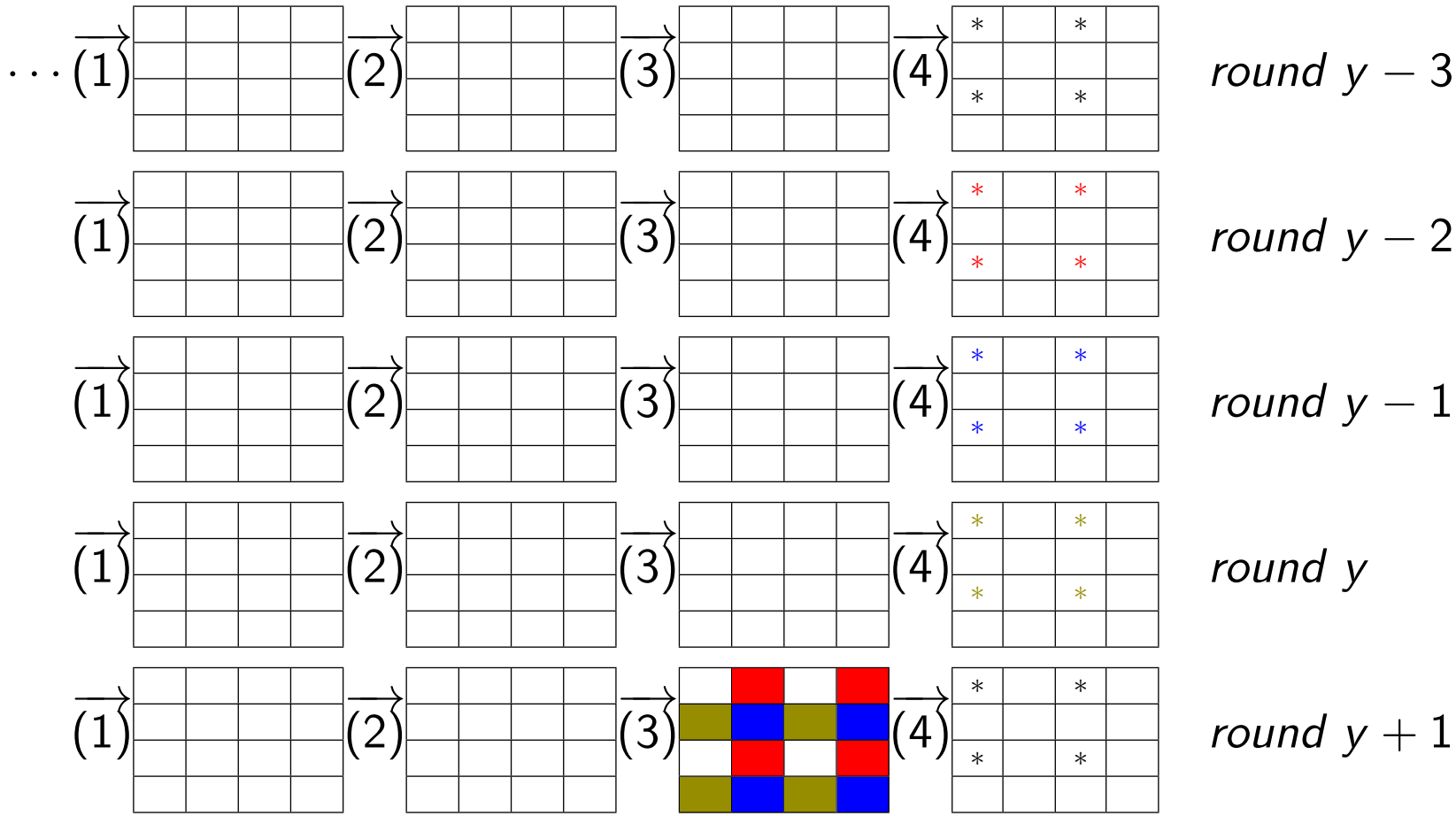
Backwards-aNd-Forwards - Step 2

Other two equation systems in the round $y+1$ are solved and the inverse transformations up to AddRoundKey in round y are performed.



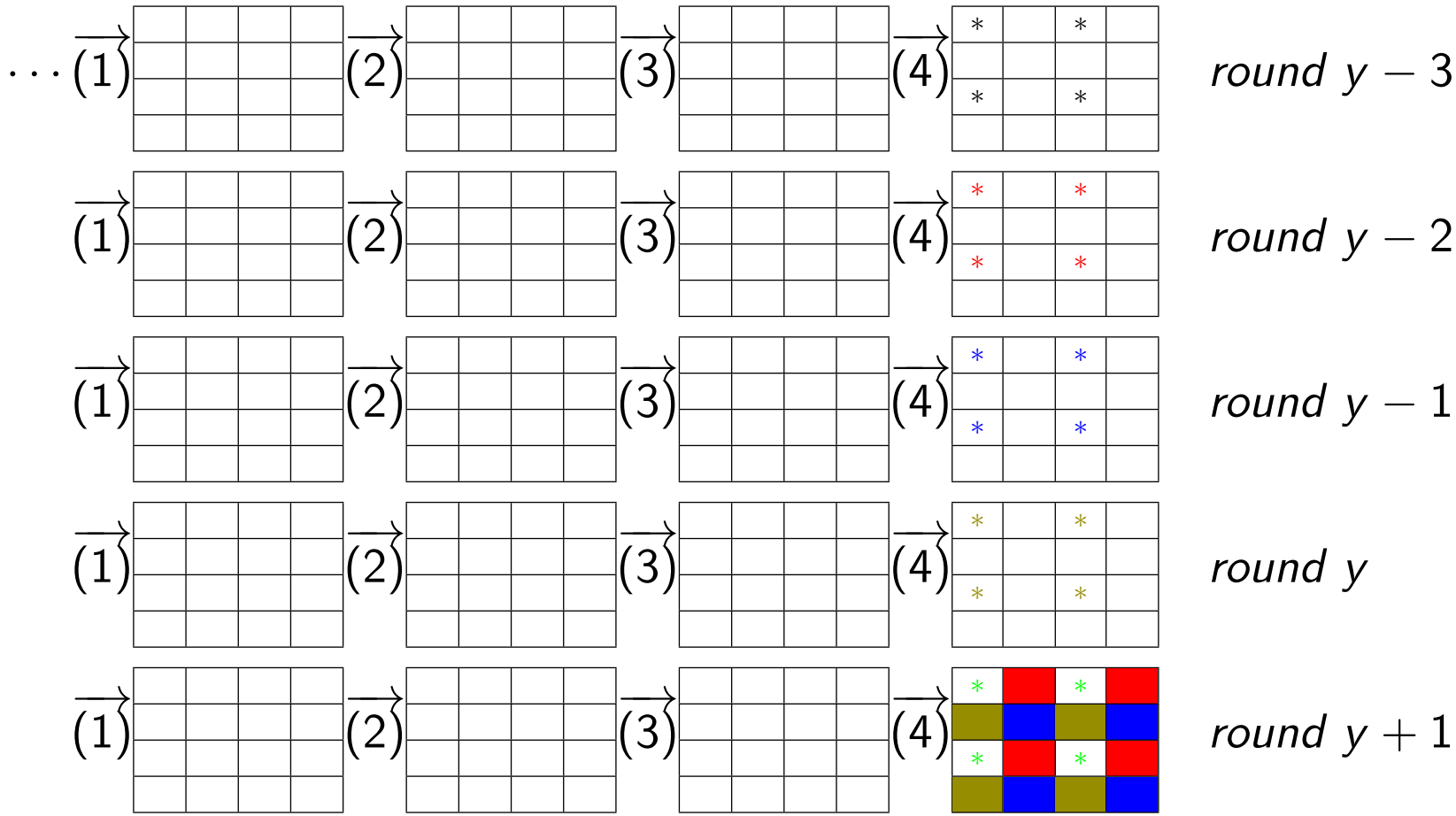
Backwards-aNd-Forwards - Step 2

The message block $\overline{x}_y$ is replaced by a collision-dependent one. At this point, both x and $\overline{x}$ collide on 96 bits after MixColumns in round $y+1$.



Backwards-aNd-Forwards - Step 3

The message block $\overline{x}_{y+1}$ is directly replaced. At this point, both x and $\overline{x}$ collide at the output of the round $y+1$.



The Attack - Hit

Sender: Computes $Tag = \text{Alpha-MAC}(x, Key)$ and sends Tag.

Attacker: Finds $\overline{x}$ such that $Tag = \text{Alpha-MAC}(\overline{x}, Key)$.

Receiver: Computes $Out = \text{Alpha-MAC}(\overline{x}, Key)$ and checks that $Out = Tag$ but the message was sent by the attacker.

The Change I - Why ShiftRows?

In the round function of Alpha-MAC, as well as in the standard AES, ShiftRows acts on the two-dimensional states cyclically rotating the rows

$\xi_{0,0}$	$\xi_{0,1}$	$\xi_{0,2}$	$\xi_{0,3}$
$\xi_{1,0}$	$\xi_{1,1}$	$\xi_{1,2}$	$\xi_{1,3}$
$\xi_{2,0}$	$\xi_{2,1}$	$\xi_{2,2}$	$\xi_{2,3}$
$\xi_{3,0}$	$\xi_{3,1}$	$\xi_{3,2}$	$\xi_{3,3}$

$\longmapsto$

$\xi_{0,0}$	$\xi_{0,1}$	$\xi_{0,2}$	$\xi_{0,3}$
$\xi_{1,1}$	$\xi_{1,2}$	$\xi_{1,3}$	$\xi_{1,0}$
$\xi_{2,2}$	$\xi_{2,3}$	$\xi_{2,0}$	$\xi_{2,1}$
$\xi_{3,3}$	$\xi_{3,0}$	$\xi_{3,1}$	$\xi_{3,2}$

Likewise, this constitutes a permutation of the one-dimensional states

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	5	10	15	4	9	14	3	8	13	2	7	12	1	6	11

called diffusion optimal permutation, since all bytes in the same column after ShiftRows belong to different columns of the previous state. Such property allows to spread the diffusion provided by MixColumns to the entire state, achieving full diffusion in two rounds.

The Change II - ShiftRows Replacements

To keep the property that **all bytes in the same column of the state belong to different columns of the previous state** the transformation ShiftRows can be replaced by any diffusion optimal permutation.

Let $\tau_0, \tau_1, \tau_2, \tau_3$ and $\mu_0, \mu_1, \mu_2, \mu_3$ be random permutations in S_4, then the transformation of the state

$$\xi_{i,j} \xrightarrow[0 \leq i,j \leq 3]{} \xi_{\mu_i[j],\tau_j[i]}$$

acts like a diffusion optimal permutation. There are exactly 24^8 possible replacements for ShiftRows.

The Change III - AES Improvements

This change preserves the strength of AES against classic attacks since the *Wide Trail Strategy* remains the same; however, 36,6 bits of extra security are provided by the uncertainty of ShiftRows.

Other AES improvements using this transformations against side channel attacks have been shown.


Spain, M. and Varia, M. Diversity Within the Rijndael Design Principles for Resistance to Differential Power Analysis. LNCS 10052, pp. 71–87, 2016.


Alfonso, A. and Freyre, P. How Secure is the Advanced Encryption Standard with Random ShiftRows against Fault Analysis? Journal of Science and Technology on Information Security. 1(07), 2018.

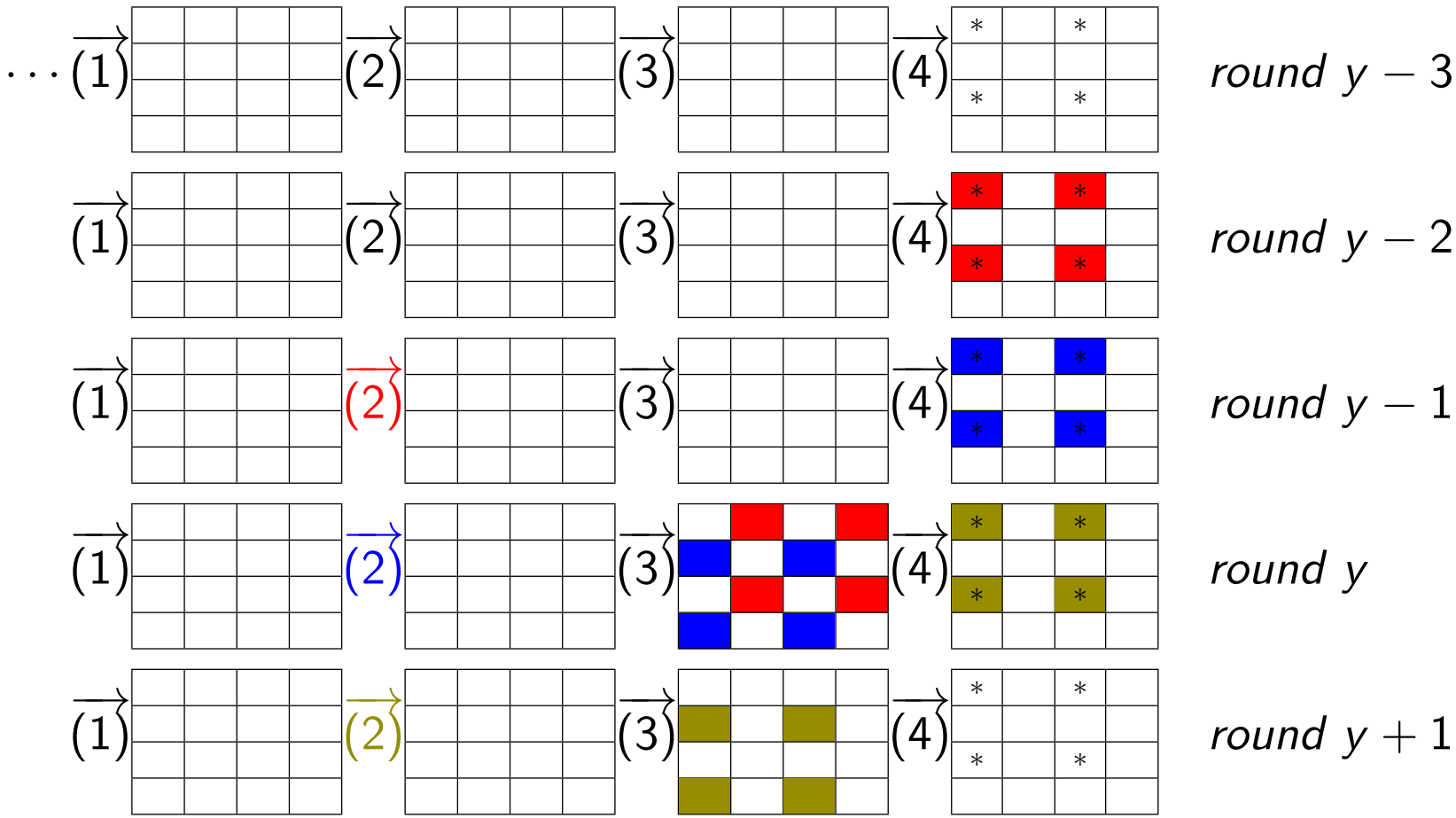
The Change IV - Improvement on Alpha-MAC

To carry out the attack the inverse of the transformation ShiftRows acts 3 times on the states.

- In the BNB search the attacker needs to move 4 collision-dependent bytes from the round y to the exactly positions of the injection at the output of the round $y-2$.
- In the BNF search the attacker needs to move 4 collision-dependent bytes from the round y to the exactly positions of the injection at the output of the round $y-1$.
- In the BNF search the attacker needs to move 4 collision-dependent bytes from the round $y+1$ to the exactly positions of the injection at the output of the round y.

In this paper we propose the replacement of ShiftRows in each unkeyed round function of Alpha-MAC by a random (unknown) diffusion optimal permutation from the set of all its possible choices.

The Change IV - Improvement on Alpha-MAC



The Change IV - Improvement on Alpha-MAC

Let's assume that the attacker is able to adapt the equation systems around MixColumns in both BNB and BNF search, then to run the full attack he assumes that the random transformations ShiftRows move the solutions (the 4 collision-dependent bytes) to the exactly positions of the injection. In this case there are only 3 possible scenarios.

- 2 bytes came from the same column of the state before MixColumns and the other 2 bytes came from another column (as in the attack).
- 2 bytes came from the same column of the state before MixColumns and the other 2 bytes came from different columns each.
- Each byte came from a different column of the state before MixColumns.

In none of the previous scenarios the order of such bytes is significant.

The Change IV - Improvement on Alpha-MAC

let's assume that the attacker adapts the equation systems so that the solutions are in the first possible scenario.

Let α and β be the columns where the solutions are located on the state before MixColumns, then:

- only 4 of the 24 possible permutations τ_α and τ_β used to construct InvShiftRows move these bytes to the columns of the injection.
- only 4 of the 24 possible permutations μ_0 and μ_2 used to construct InvShiftRows move these bytes to the rows of the injection.

The remaining permutations used to construct InvShiftRows do not influence this movement.

The Change IV - Improvement on Alpha-MAC

let's assume that the attacker adapts the equation systems so that the solutions are in the second possible scenario.

Let α, β and γ be the columns where the solutions are located on the state before MixColumns, then:

- only 4 of the 24 possible permutations τ_α used to construct InvShiftRows move the 2 bytes to the columns of the injection.
- only 12 of the 24 possible permutations τ_β used to construct InvShiftRows move the byte to one of the columns of the injection.
- only 6 of the 24 possible permutations τ_γ used to construct InvShiftRows move the byte to the another column of the injection.
- only 4 of the 24 possible permutations μ_0 and μ_2 used to construct InvShiftRows move these bytes to the rows of the injection.

The remaining permutations used to construct InvShiftRows do not influence this movement.

The Change IV - Improvement on Alpha-MAC

let's assume that the attacker adapts the equation systems so that the solutions are in the third possible scenario.

- only 12 of the 24 possible permutations τ_0 used to construct InvShiftRows move the byte to one of the columns of the injection.
- only 12 of the 24 possible permutations τ_1 used to construct InvShiftRows move the byte to one of the columns of the injection.
- only 6 or 12 of the 24 possible permutations τ_2 used to construct InvShiftRows move the byte to one of the columns of the injection, depending if τ_0 and τ_1 move the bytes to the same column or not.
- only 6 of the 24 possible permutations τ_3 used to construct InvShiftRows move the byte to the another column of the injection.
- only 4 of the 24 possible permutations μ_0 and μ_2 used to construct InvShiftRows move these bytes to the rows of the injection.

The remaining permutations used to construct InvShiftRows do not influence this movement.

The Change IV - Improvement on Alpha-MAC

Let us denote by ρ the probability that the solutions were moved to the positions of the injection through the random transformation ShiftRows, then the success probability of the attack is ρ^3.

- In the first possible scenario

$$\rho = \frac{4^4}{24^4} = \frac{1}{2^4(2+1)^4} = \frac{1}{2^4(2^6+17)} = \frac{1}{2^{10}+\varepsilon_1} < 2^{-10}$$

- In the second possible scenario

$$\rho = \frac{4^3 \cdot 12 \cdot 6}{24^5} = \frac{1}{2^6(2+1)^3} = \frac{1}{2^6(2^4+11)} = \frac{1}{2^{10}+\varepsilon_2} < 2^{-10}$$

- In the third possible scenario

$$\rho = \frac{4^2 \cdot (6 \cdot 6 + 6 \cdot 12) \cdot 6}{24^6} = \frac{1}{2^9(2+1)} = \frac{1}{2^{10}+\varepsilon_3} < 2^{-10}$$

An upper bound for the success probability of the full attack is 2^{-30}

Further Comments - I

Permutations τ_i and μ_i for all $0 \leq i \leq 3$ must remain unknown in all rounds to the attacker even when he knows the intermediate state or the secret key, therefore such permutations should depend on pseudo-random sequences independent of the secret key. A possible solution is to generate as many pseudo-random sequences as necessary from a seed using the key schedule of AES.

Further Comments - II

We also recommend the replacement of the fixed MDS matrix of MixColumns by another randomly generated from the set of all its possible choices.

Freyre, P. et al. Random Generation of MDS Matrices. Third Workshop on Current Trends in Cryptology, pp. 105–114, 2014.

With this change the entries of the MDS matrix are unknown to the attacker and he can't solve the equation systems around MixColumns even if he knows the intermediate state or the secret key.

Conclusion

- Alpha-MAC is an AES-based instance of the construction Alred for MAC functions using the round function of AES as cryptographic primitive where the round keys are replaced by message injections.
- Huang, Seberry and Susilo presented an attack where it is only necessary to know an intermediate state (or the secret key) and 5 message blocks (with 32 bits each) to find a second preimage.
- Replacing ShiftRows by a random (unknown) diffusion optimal permutation in each unkeyed round of Alpha-MAC the full attack has a success probability less than 2^{-30}.
- The replacement of ShiftRows and MixColumns is also proposed for the keyed round function of the AES step at the beginning and the end of the Alpha-MAC algorithm. It can break other kind of attacks on Alpha-MAC to recover the intermediate state or the secret key.

Thank you for your attention !!!

Extending AES Improvements: A Proposal for Alpha-MAC in View of Collision Resistance

Adrián Alfonso Peñate and Pablo Freyre Arrozarena
Institute of Cryptography, University of Havana, Cuba

9th Workshop on Current Trends in Cryptology
September 15-17, 2020