A Guidance of The Full-Fledged Model

Overview

Infrastructure

To run the model, ensure "Minizinc with Or-Tools" is installed on your computing device.

Note that, we run the model on *AMD Ryzen 9 7945HX* with *32* threads, using:

```
minizinc --solver CP-SAT -p 32 Modelname.mzn > RESULTs.txt
```

Instruction manual

All variables in our model (*solver = CP-SAT*) are integers; when floats arise, try to transform them to a integer if it`s possible.

Components

Each part of our model is separated in different `dzn` files, therefore, we have the following components:

- **Main:** `0_Deoxys. mzn` is the center control model. By enabling or disabling the functions in this main model, it`s possible to adjust the searching depth.
- **Pattern (Dis. and Ext.):** "1_differential.dzn" constraints the differential pattern of the whole attack.
  - Returning parameters: #active cells in plaintext, ciphertext r_b, r_f, #involved subkeys m_b, m_f, and $probability$.
  - The probabilistic extension is allowed, and m_b, m_f can be smaller when the state test is working in this component.
  - Open `include "1_differential.dzn";` to use this function.
- **Guessing Strategy:** "2_guess_and_determine.dzn" describes the pre-guessing strategy of subkeys.
  - Returning parameters: pre-guessed subkeys m'_b, m'_f, filters r'_b, r'_f.
  - $Gstate$ is also contain in m'_b, m'_f when the state test is working.
  - Open `include "2_guess_and_determine.dzn";` to use this function.
- **Epsilon calculation:** "3-1_epsilon_gandf-ST.dzn" and "3-2_epsilon_table-ST.dzn" give the constraints of step assignment and complexities.
  - Returning parameters: processing step of the unguessed subkeys, corresponding filters, time and memory complexity of each step.
  - When the state test is working, ST is contained as subkeys, and EGK is eliminated, in some steps.
  - Open `include "3-1_epsilon_gandf-ST.dzn";` to use the guess-and-filter approach, and `include "3-2_epsilon_table-ST.dzn";` for the pre-computation hash table approach.

Predicates: "predicates.dzn"

- `include "3-1_epsilon_gandf-ST.dzn";` is set as default.
- Many predicates are included as methods that can be used in other components.

Adjustments in Main

State test

The switch of state test:

```
int: OpenST = 0; % 1 -> Trun on State Test;
constraint if OpenST = 0
          then forall(r in 0..Rb-1, i in 0..15)(uST[r,i] = -1) /\ forall(r in
Rb+Ru+Rm+Rl..Rb+Ru+Rm+Rl+Rf-1, i in 0..15)(lST[r,i] = -1) /\
                forall(r in 0..Rb-1, i in 0..15)(uEGK[r,i] = -1) /\
                forall(r in 0..Rb-1, i in 0..15)(uESB[r,i] = -1) /\
                forall(r in 0..Rb-1, i in 0..15)(uEMC[r,i] = -1) /\
                forall(r in Rb+Ru+Rm+Rl+1..Rb+Ru+Rm+Rl+Rf, i in 0..15)(lEGK[r,i]
= -1) /\
                forall(r in Rb+Ru+Rm+Rl..Rb+Ru+Rm+Rl+Rf-1, i in 0..15)(lESB[r,i]
= -1) /\
                forall(r in Rb+Ru+Rm+Rl..Rb+Ru+Rm+Rl+Rf-2, i in 0..15)(lEMC[r,i]
= -1)
          endif;
```

epsilon calculation

Switch between two approaches of epsilon calculation:

```
% Part 3 (epsilon G and F)
% include "3-1_epsilon_gandf-ST.dzn";

% Part 3 (epsilon table)
include "3-2_epsilon_table-ST.dzn";
```

Key bridging

The following constraints of strong key bridging are applied only when an attack achieves no less than 15 rounds.

3 cases for involvements, guessing strategy, and epsilon calculation....

```
% CASE-1: Strong Key Bridges for involved key
array[0..3] of var int: vRd;
constraint forall(c in 0..3)(vRd[c] = JTable[sum(i in 0..3)
(uVSTK[0,hTable[4*c+i,1]]), sum(i in 0..3)(lVSTK[15,4*c+i])]);
var int: mb  = sum(r in 0..Rb-1, i in 0..15)(uVSTK[r,i] == 1 /\ uEGK[r,i] == -1)
+ sum(r in 0..Rb-1, i in 0..15)(uVstate[r,i]);
var int: mf  = sum(r in Rb+Ru+Rm+Rl+1..Rb+Ru+Rm+Rl+Rf, i in 0..15)(lVSTK[r,i] ==
1 /\ lEGK[r,i] == -1) + sum(r in Rb+Ru+Rm+Rl..Rb+Ru+Rm+Rl+Rf-1, i in 0..15)
(lVstate[r,i]);
var int: mk = 8*(mb + mf) - sum(c in 0..3)(vRd[c]);

% Strong Key Bridges for pre-guessed key
array[0..3] of var int: gRd;
constraint forall(c in 0..3)(
```

```
    if forall(i in 0..3)(uGSTK[0,hTable[4*c+i,1]] = uVSTK[0,hTable[4*c+i,1]] /\
lGSTK[15,4*c+i] = lVSTK[15,4*c+i])
    then gRd[c] = JTable[sum(i in 0..3)(uGSTK[0,hTable[4*c+i,1]]), sum(i in 0..3)
(lGSTK[15,4*c+i])]
    else gRd[c] = 0
    endif);
var int: mb_p = sum(r in 0..Rb-1, i in 0..15)(uGSTK[r,i]) + sum(r in 0..Rb-1, i
in 0..15)(uGstate[r,i]);
var int: mf_p = sum(r in Rb+Ru+Rm+Rl+1..Rb+Ru+Rm+Rl+Rf, i in 0..15)(lGSTK[r,i]) +
sum(r in Rb+Ru+Rm+Rl..Rb+Ru+Rm+Rl+Rf-1, i in 0..15)(lGstate[r,i]);
var int: mk_p = 8*(mb_p + mf_p) - sum(c in 0..3)(gRd[c]);

% Strong Key Bridges for epsilon calculation
array[0..3] of var -1..Step: sBigRd;
array[0..3] of var int: sRd;
constraint forall(c in 0..3)(sBigRd[c] = max(max(i in 0..3)(uSGK[0, hTable[4*c+i,
1]]), max(i in 0..3)(lSGK[15, 4*c+i])));
constraint forall(c in 0..3)(
    if sBigRd[c] >= 1
    then sRd[c] = JTable[sum(i in 0..3)(uSGK[0,hTable[4*c+i,1]] >= 1 /\
uSGK[0,hTable[4*c+i,1]] <= sBigRd[c]),
                         sum(i in 0..3)(lSGK[15,4*c+i] >= 1 /\ lSGK[15,4*c+i] <=
sBigRd[c])]
    else sRd[c] = 0
    endif);
array[1..Step] of var int: sRdS;
constraint forall(s in 1..Step)(if exists(c in 0..3)(sBigRd[c] = s) then sRdS[s]
>= 0 else sRdS[s] = 0 endif);
constraint forall(c in 0..3)(if sBigRd[c] >= 1 then sRdS[sBigRd[c]] = sRd[c]
endif);
```

We give the format output to show the influence of key bridging (redundancy in different processes): easy to verify where the key bridging is working.

```
output[show(mb+mf) ++ " bytes |" ++ show(sum(c in 0..3)(vRd[c])) ++ " bits ------
"]; % involvement
output[show(mb_p+mf_p) ++ " bytes |" ++ show(sum(c in 0..3)(gRd[c])) ++ " bits --
---- "]; % pre-guessing
output[show([sRdS[s] | s in 1..Step])]; % epsilon calculation
```

Objective Functions for Elasticizing

Note that the extra variables for disabling functions exist throughout the entire model, but are constrained out (assigning constants) of solutions when they provide influences.

Main + Pattern

$$\min(a \times PrAtt + b \times (m_b + m_f))$$

Main + Pattern+ Guessing Strategy

$$constraint \ Time_\epsilon = 0$$
$$\min(a \times Time + b \times Memory + c \times Date)$$

Whole Model

$$\min(a \times Time + b \times Memory + c \times Memory_\epsilon + d \times Data)$$

Due to the long time required to solve the whole model, we provide some constraints as *TEST* for fast verification.

The input of our model

Alter-abling specifications for different attacks (example = 11-round rectangle attack on `Deoxys-BC-256`)

```
int: SpecDeoxys = 2; % denotes TK2
int: block_size = 128;
int: key_size = block_size * SpecDeoxys;

int: Rb = 1; % backward extension
int: Ru = 3; % upper differential
int: Rm = 2; % middle
int: Rl = 3; % lower differential
int: Rf = 2; % forward extension
```

Examples

We give the models and the corresponding results of 2 versions of `Deoxys`, goto:

- **11-round** `Deoxys-BC-256`
- **15-round** `Deoxys-BC-384`

We give the models and the corresponding results of 2 versions of `Deoxys-AE`, goto:

- **10-round** `Deoxys-I-128-128`
- **14-round** `Deoxys-I-256-128`

We give the models and the corresponding results of 2 versions of `Deoxys-multiTK`, goto:

- **16-round** `Deoxys-TBC-512-256`
- **18-round** `Deoxys-TBC-640-256`

Deoxys-BC-256 (11r)

```
int: SpecDeoxys = 2;
int: block_size = 128;
int: key_size = block_size * SpecDeoxys;

int: Rb = 1;
int: Ru = 3;
int: Rm = 2;
int: Rl = 3;
int: Rf = 2;
```

Result obtained (detail):

```
Complexity of Rectangle Attack on Deoxys-BC-384:

Parameters:
---------------------
rb= 4 | rb'= 4 | mb= 4 | mb'= 4
rf=12 | rf'= 4 | mf=18 | mf'= 5
---------------------
Prob.  : 112
Data   : 122
MemoryC: 123
TimeC  : 195 [T1=194 | T2u=195, T2l=253 | T3=188 (epsilon=0)]
```
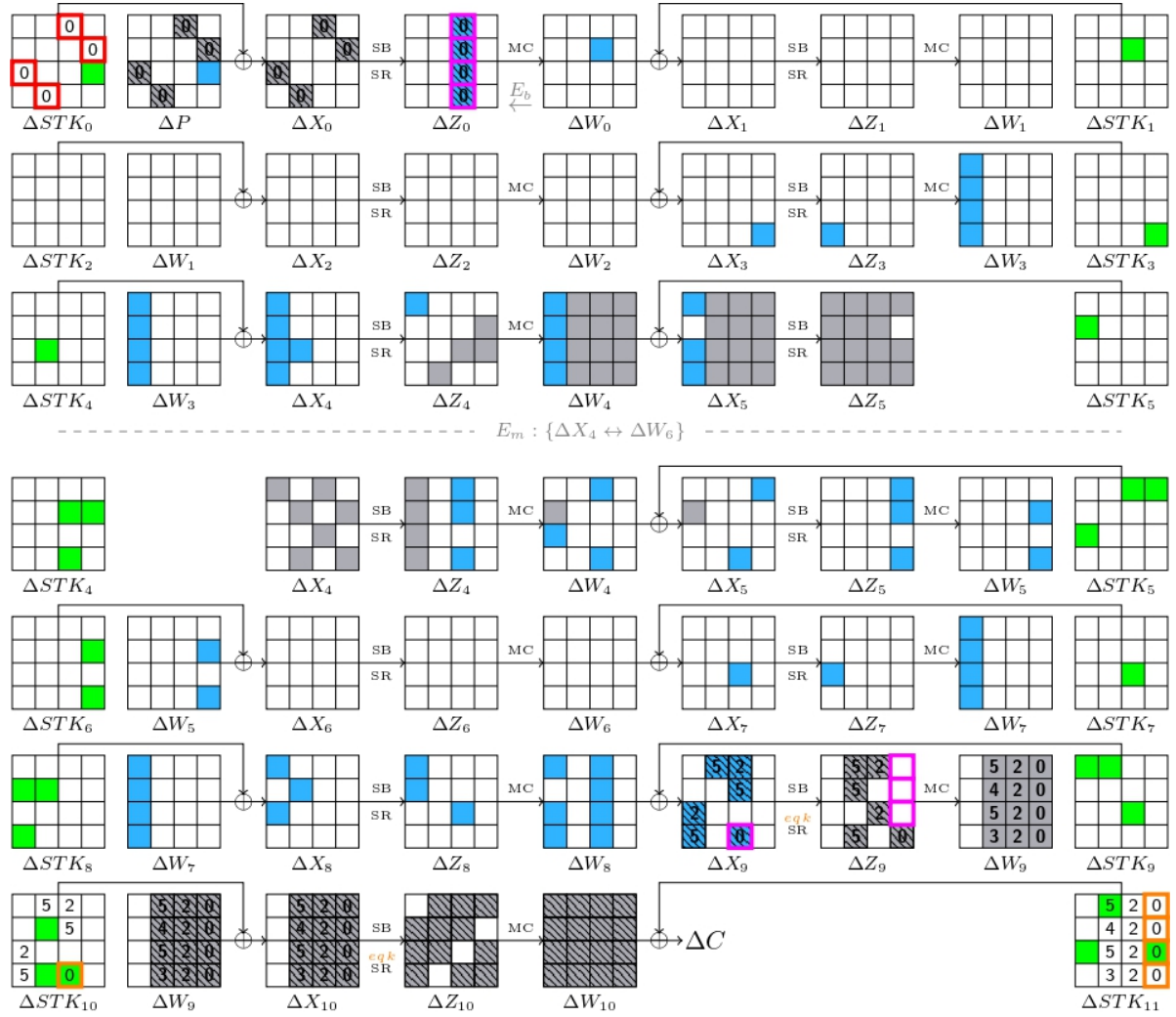
Draw with TikZ latex:

Prob.: 112 (0, 14, 42, 56, 0); $r_b = 4$ $r'_b = 4$ $m_b = 4$ $m'_b = 4$; $r_f = 12$ $r'_f = 4$ $m_f = 18$ $m'_f = 5$
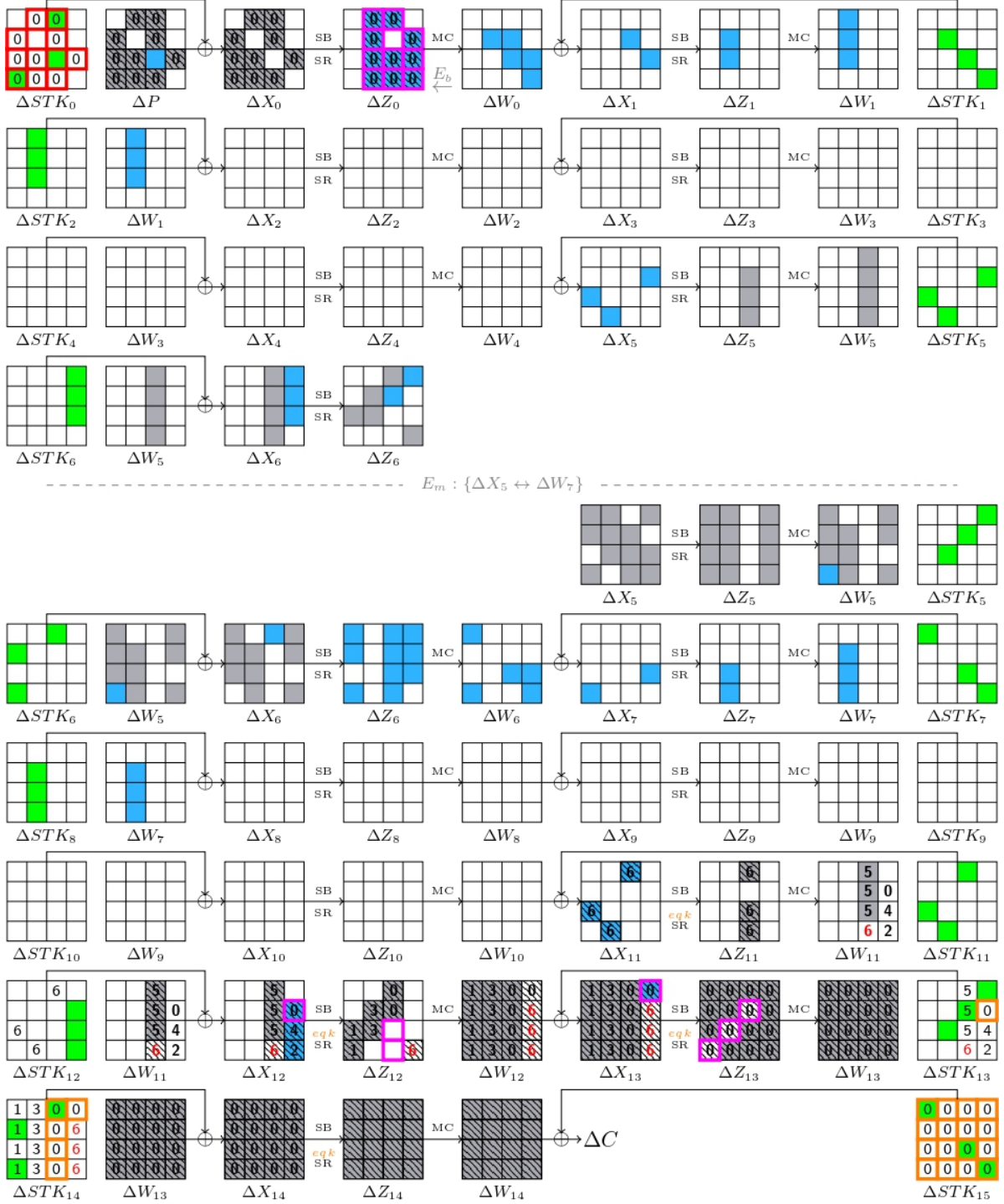DataC: 122, MemoryC: 123 ($M_\epsilon = 14$), TimeC: 195 [T_1=194, T_{2u}=195, T_3=188 (T_ϵ=0)]

Deoxys-BC-384 (15r)

```
int: SpecDeoxys = 3;
int: block_size = 128;
int: key_size = block_size * SpecDeoxys;

int: Rb = 1;
int: Ru = 4;
int: Rm = 2;
int: Rl = 4;
int: Rf = 4;
```
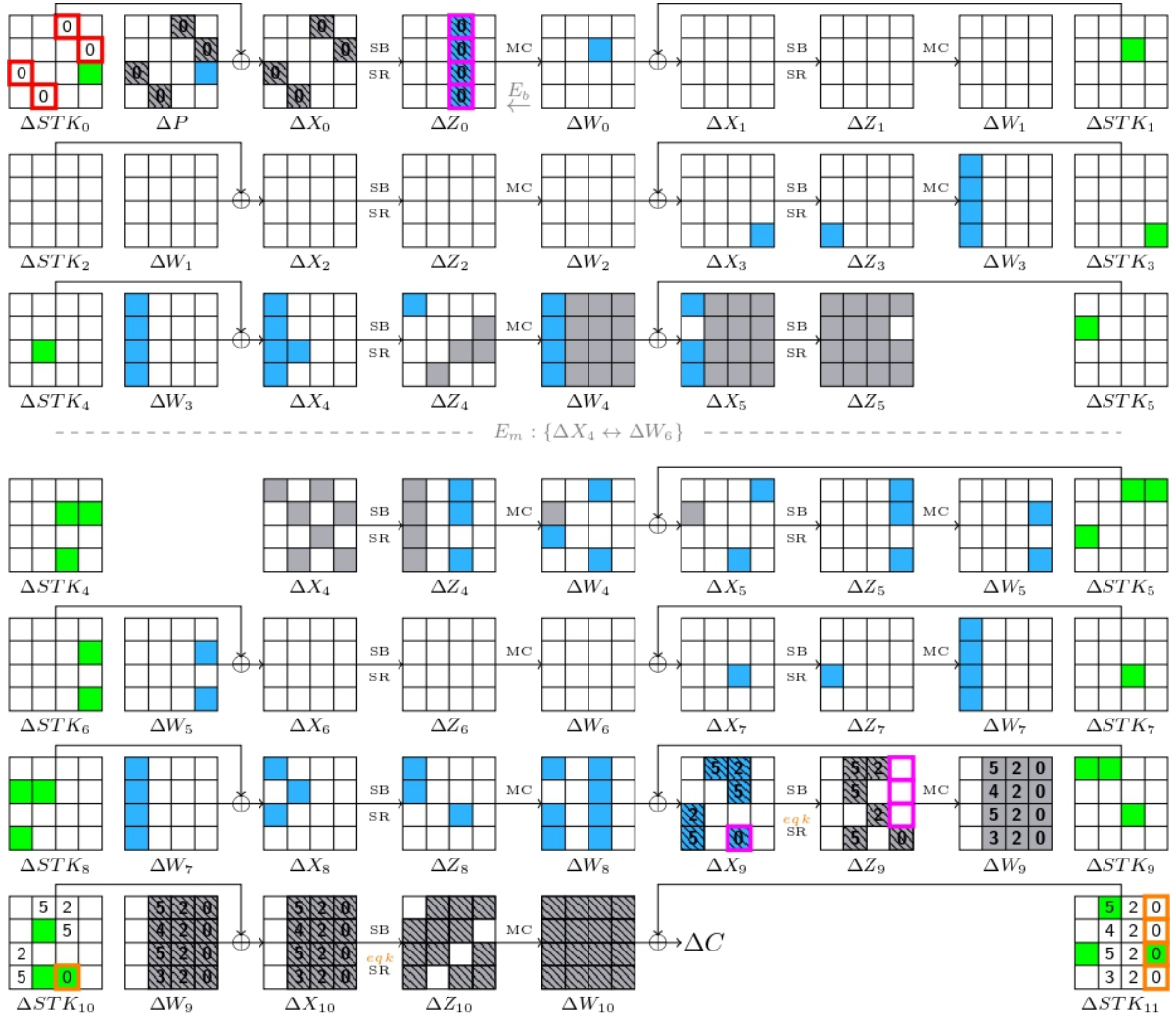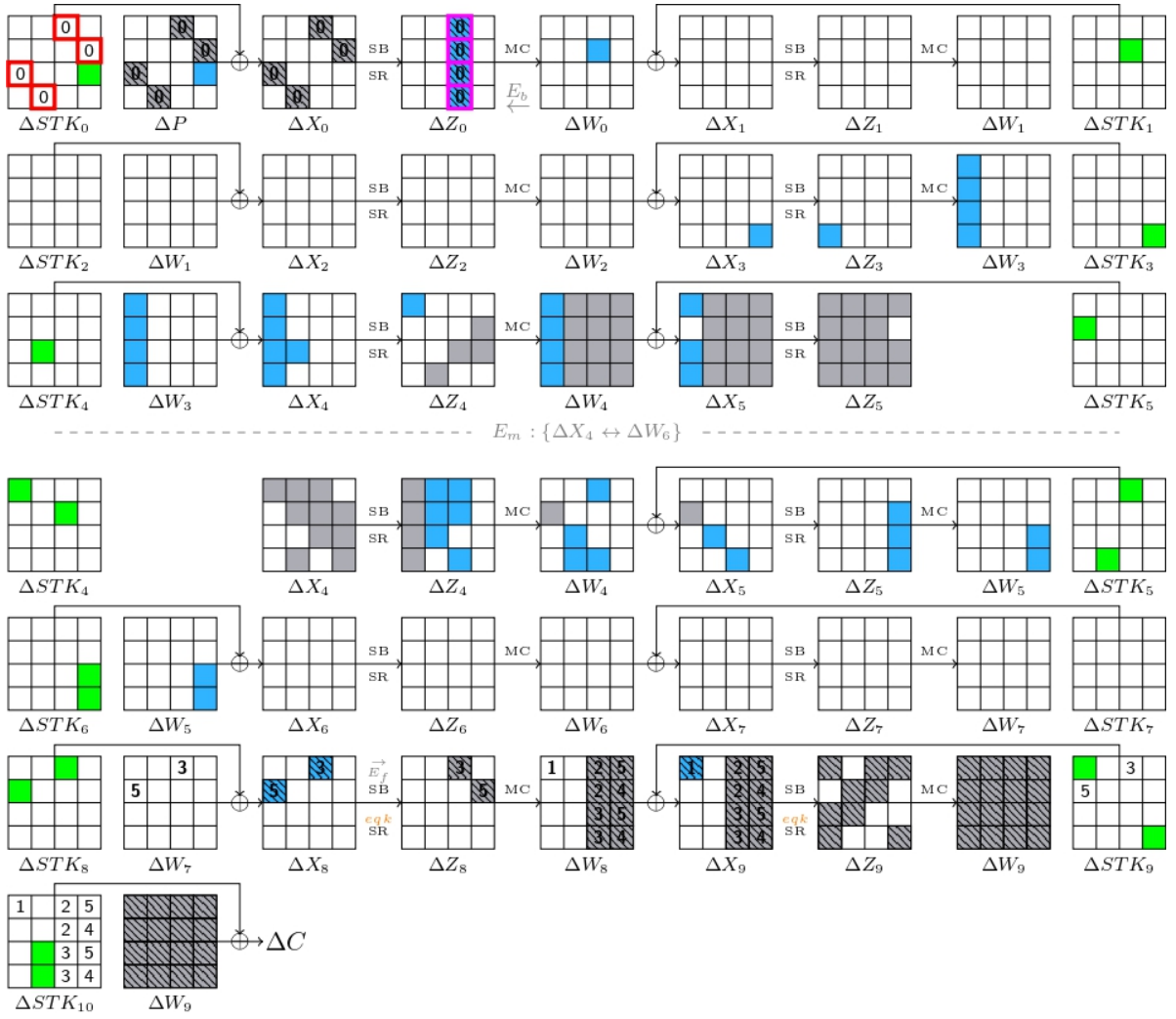

$E_m : \{\Delta X_5 \leftrightarrow \Delta W_7\}$

Prob.: 98 (0, 28, 26, 28, 16); $r_b = 10$ $r'_b = 10$ $m_b = 10$ $m'_b = 10$; $r_f = 16$ $r'_f = 7$ $m_f = 38$ $m'_f = 22$

DataC: 115, MemoryC: 128 ($M_\epsilon = 16$), TimeC: 334 [T_1=331, T_{2u}=332, T_3=334 (T_ϵ=0)]
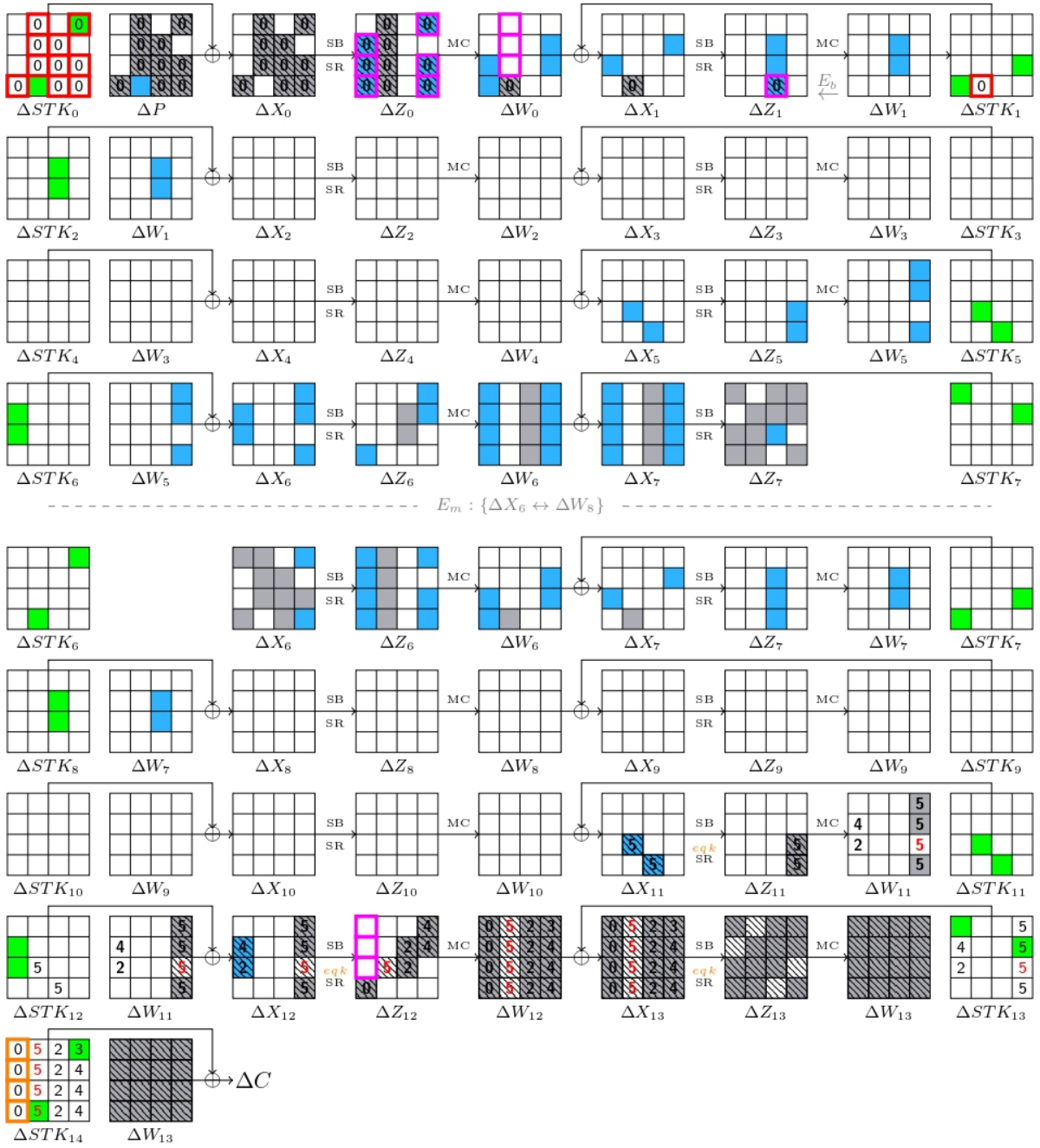
Deoxys-BC-256 (11r)

Prob.: 112 (0, 14, 42, 56, 0); $r_b = 4$ $r'_b = 4$ $m_b = 4$ $m'_b = 4$; $r_f = 12$ $r'_f = 4$ $m_f = 18$ $m'_f = 5$

DataC: 122, MemoryC: 123 ($M_\epsilon = 14$), TimeC: 195 [$T_1 = 194$, $T_{2u} = 195$, $T_3 = 188$ ($T_\epsilon = 0$)]

Deoxys-I-128 (10r)

Prob.: 56 (0, 14, 42, 0, 0); $r_b = 4$ $r'_b = 4$ $m_b = 4$ $m'_b = 4$; $r_f = 9$ $r'_f = 0$ $m_f = 11$ $m'_f = 0$
DataC: 94, MemoryC: 95 ($M_\epsilon = 6$), TimeC: 127 [$T_1 = 126$, $T_{2u} = 127$, $T_3 = 124$ ($T_\epsilon = 2$)]

Deoxys-I-256 (14r)

Prob.: 102 (28, 28, 30, 0, 16); $r_b = 10$ $r'_b = 10$ $m_b = 11$ $m'_b = 11$; $r_f = 12$ $r'_f = 3$ $m_f = 19$ $m'_f = 4$

DataC: 117, MemoryC: 120 ($M_\epsilon = 15$), TimeC: 242 [$T_1 = 237$, $T_{2u} = 238$, $T_3 = 242$ ($T_\epsilon = 0$)]

Pretty Output, Drawing Using TikZ

```
include "./format_out/1_format_out_diff.dzn"; % show differential pattern,
parameters, and probability
include "./format_out/3-2_format_out_tableST.dzn"; % show step assignment and
complexities of epsilon
include "./format_out/0_format_out_Main.dzn"; % show the necessary parameters,
complexities of the attack
include "./format_out/drawlatex.dzn"; % draw with tikz
```

Copy the LaTeX code output and paste it into a LaTeX compiler.

For SKINNY

`SKINNY` is a typical example that uses a non-MDS matrix as its linear layer; therefore, in the key recovery phase, differential-determination detection beyond value determination is necessary in the components Guessing Strategy and Epsilon Calculation.

For each differential of state, `x`, sets of variables we used in the key recovery model of `SKINNY`.

```
vX: value needed
dX: differential needed
gSTK: guessed subkey in Guessing Strategy
detX: value determined
detdiffX: differential determined
sgSTK: guessed subkey in step assignment
saX: value assigned step
sdvX: value deduced in step assignment
```

Considering new sets of variables, `detdiffX, sadiffX, sadedX, sadevX`, allows more delicate relations to be captured in our model. Specifically, in the Epsilon Calculation component, the **Sbox property** can be used to deduce subkeys that are not guessed in any process.

```
sadiffX: differential assigned step
sadedX: differential deduced using the property of Sbox property
sadevX: value deduced using the property of Sbox property
```

An example for deducing `sadedX,sadevX`, where `X, Y, W` represent the state before Sbox, after Sbox, and before MC, respectively:

```
(W[col.2] = X[col.0] + X[col.3])
if dW[col.2] = 0 (difference of W)
then
    sadedX[col.0] = max(detdiffX[col.3], sadiffX[col.3], sadedX[col.3])
    sadedX[col.3] = max(detdiffX[col.0], sadiffX[col.0], sadedX[col.0])
if sadedX = step and sadedY <= step
then sadevX = step
```

The stronger key bridging:

The key bridging component should be adapted to the number of attack rounds. When the attack covers more than 30 rounds, there are subkeys involved, and a full round can be deduced at no cost. To capture all involved subkeys, we use *sliding windows* that cover 30 rounds.

For example, for the 33-round attack, the extended rounds span in rounds 0-3, 27-33. The key bridging component relies on the five slide windows that cover the outer rounds: (0 .. 3, 27 .. 29), (1.. 3, 27 .. 30), (2 .. 3, 27 .. 31), (3 .. 3, 27 .. 32).

For classical key bridging, the subkeys in the same LANE provide at most p-cell information, which can be easily embedded into the component of stronger key bridging, where p is the number of `TK` ($p = 3$ for SKINNY-n-3n).

```
constraint forall(c in 0..15)(
    let {
        var int: vsum029 = sum(r in 0..4)(bvSTK[r,c]) + sum(r in 27..29)
(fvSTK[r,c]),
        var int: vsum130 = sum(r in 1..4)(bvSTK[r,c]) + sum(r in 27..30)
(fvSTK[r,c]),
        var int: vsum231 = sum(r in 2..4)(bvSTK[r,c]) + sum(r in 27..31)
(fvSTK[r,c]),
        var int: vsum332 = sum(r in 3..4)(bvSTK[r,c]) + sum(r in 27..32)
(fvSTK[r,c]),
        var int: vsum433 = sum(r in 4..4)(bvSTK[r,c]) + sum(r in 27..33)
(fvSTK[r,c]),
    } in (
        vL_bg[c] = max(max(vsum029, vsum130), max(max(vsum231, vsum332),
vsum433))
    )
);
vSTK_Sbg = sum(c in 0..15)(if vL_bg[c] >= 3 then 3 else vL_bg[c] endif);
```
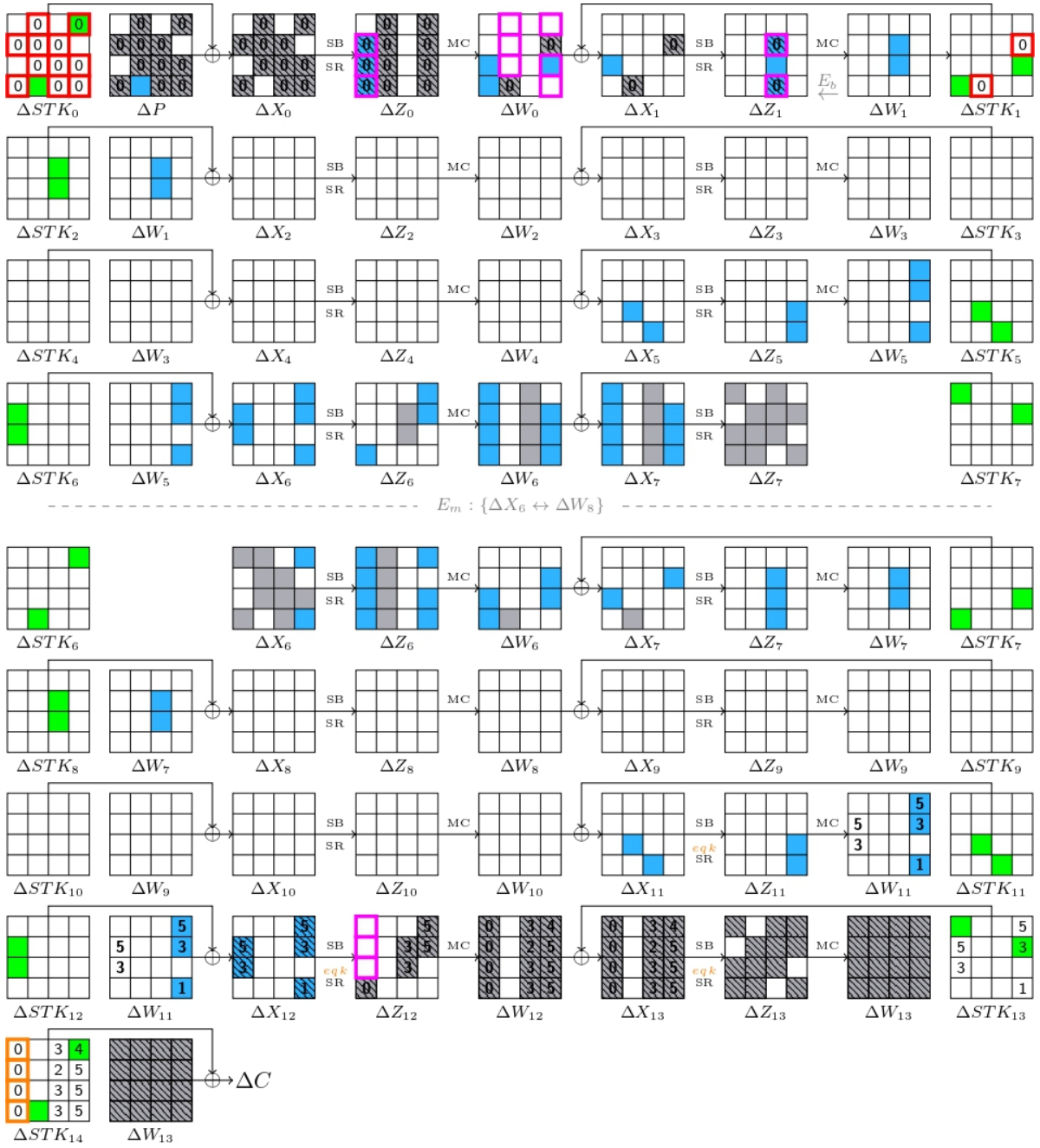
Comparison

We provide the comparison of the effect among different approaches, specifically, the state test and epsilon calculation that impact the final complexities.

State Test:

We take the 14-round attack on `Deoxys-I-256` as an example.

Under the same key-difference pattern, when closing the state test, we obtain an optimal solution whose time complexity is significantly higher than when the state test is employed.
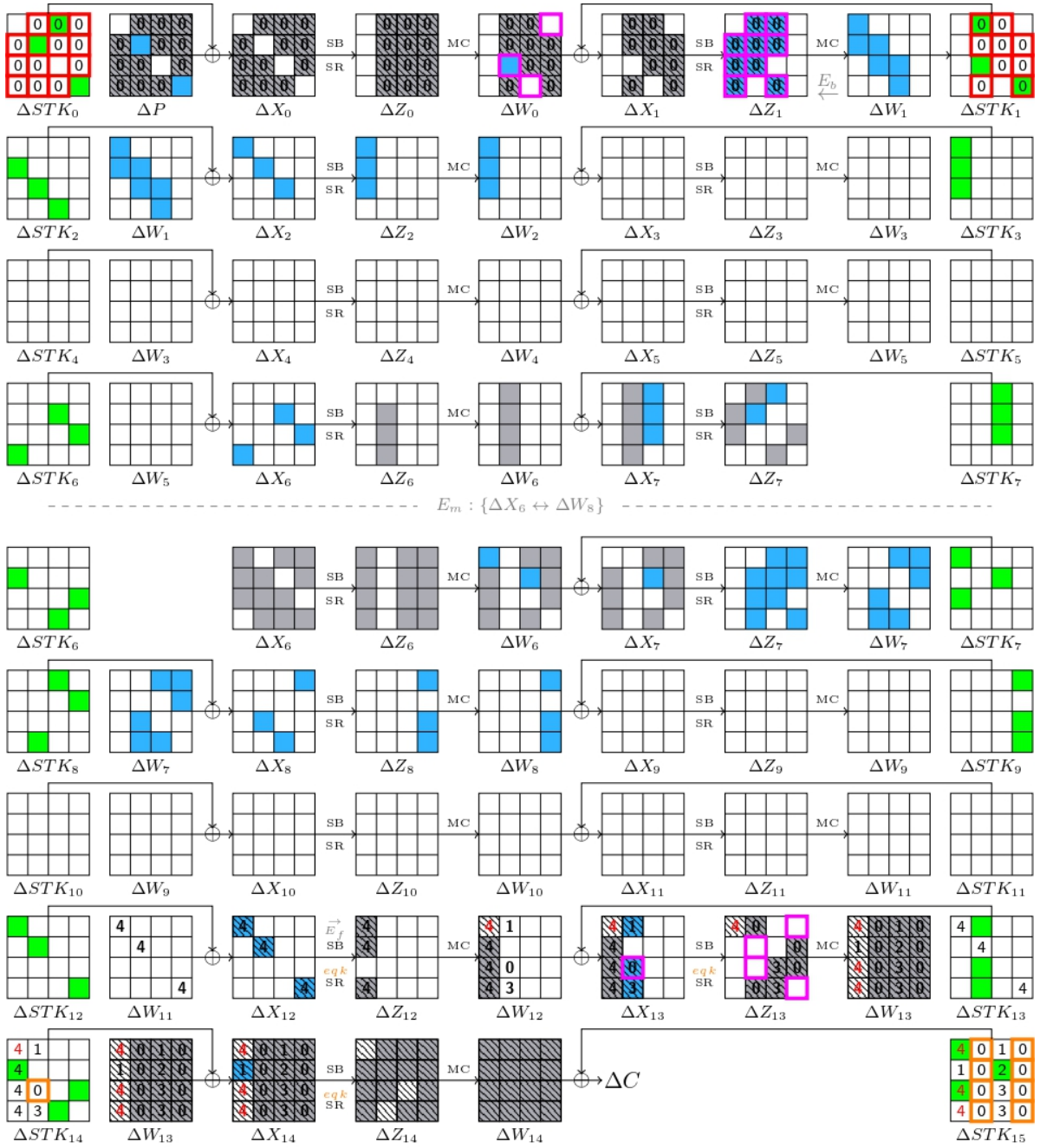
Prob.: 100 (14, 28, 30, 0, 28); $r_b = 11$ $r_b' = 11$ $m_b = 13$ $m_b' = 13$; $r_f = 12$ $r_f' = 3$ $m_f = 17$ $m_f' = 4$

DataC: 116, MemoryC: 117 ($M_\epsilon = 13$), TimeC: 256 [$T_1 = 252$, $T_{2u} = 253$, $T_3 = 256$ ($T_\epsilon = 0$)]

Epsilon Calculation:

We take the 11-round attack on `Deoxys-I-128` as an example:

Consider the multi-objective optimize

In many cases, when we optimize time complexity alone, memory complexity can exceed expectations. Therefore, the multi-objective optimization is necessary for solving, and the results we obtained that are exhibited in our paper are selected considering the time, data, and memory complexity together.

We provide a pattern that corresponds to the alternative results of the 15-round `Deoxys-BC-384`. For this pattern, the time complexity decreases slightly, whereas the *data complexity increases*.

Prob.: 112 (0, 42, 12, 42, 16); $r_b = 12$ $r_b' = 12$ $m_b = 21$ $m_b' = 21$; $r_f = 13$ $r_f' = 5$ $m_f = 22$ $m_f' = 9$
DataC: 122, MemoryC: 122 ($M_\epsilon = 13$), TimeC: 331 [$T_1 = 330$, $T_{2u} = 331$, $T_3 = 324$ ($T_\epsilon = 0$)]

Search efficiency

We provide five models for the attack instances outlined in our paper, along with the time required to solve each model. To enable fast verification, each submitted model is equipped with pattern constraints.

Searching Strategy

Solving the optimal.

Time for lower bound searching:

- 10-round `Deoxys-I-128`: in 5 minutes
- 11-round `Deoxys-BC-256`: in 10 minutes
- 14-round `Deoxys-I-256`: in 9 hours
- 15-round `Deoxys-BC-384`: in 1 day

Time for finding the optimal solution (under the constraints given in the submitted models):

- 10-round `Deoxys-I-128`: in 10 seconds
- 11-round `Deoxys-BC-256`: in 10 seconds
- 14-round `Deoxys-I-256`: in 1 minute
- 15-round `Deoxys-BC-384`: in 1 miunte

- key recovery phase of SKINNY: in 1 minute