SCIENCE CHINA
Information Sciences

• **Supplementary File** •

A smaller first-order DPA resistant AES implementation with no fresh randomness

Man WEI[1,2,3], Siwei SUN[1,2,3*], Zihao WEI[1,2,3], Zheng GONG[4] & Lei HU[1,2,3]

[1]*State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing* 100093*, China;*
[2]*Data Assurance and Communication Security Research Center, Chinese Academy of Sciences, Beijing* 100093*, China;*
[3]*School of Cyber Security, University of Chinese Academy of Sciences, Beijing* 100049*, China;*
[4]*School of Compute Science, South China Normal University, Guangzhou* 510631*, China*

Appendix A Searching for a uniform sharing of the $\mathbb{F}_{2^4}$ multiplier and the $\mathbb{F}_{2^4}$ square scaler

The function **Muls** can be represented as a vectorial Boolean function:

$$\begin{aligned}
(y^1,y^2,y^3,y^4) =& Muls(x^1,x^2,x^3,x^4,x^5,x^6,x^7,x^8)\\
=&(x^1,x^2,x^3,x^4)\times(x^5,x^6,x^7,x^8)+Sq.Sc(x^1+x^5,x^2+x^6,x^3+x^7,x^4+x^8)\\
y^1 =&x^1x^5+x^3x^5+x^4x^5+x^2x^6+x^3x^6+x^1x^7+x^2x^7+x^3x^7\\
&+x^4x^7+x^1x^8+x^3x^8+x^2+x^6+x^4+x^8;\\
y^2 =&x^2x^5+x^3x^5+x^1x^6+x^2x^6+x^4x^6+x^1x^7+x^3x^7+x^2x^8\\
&+x^4x^8+x^1+x^5+x^3+x^7;\\
y^3 =&x^1x^5+x^2x^5+x^3x^5+x^4x^5+x^1x^6+x^3x^6+x^1x^7+x^2x^7\\
&+x^3x^7+x^1x^8+x^4x^8+x^3+x^7+x^4+x^8;\\
y^4 =&x^1x^5+x^3x^5+x^2x^6+x^4x^6+x^1x^7+x^4x^7+x^2x^8+x^3x^8\\
&+x^4x^8+x^4+x^8.
\end{aligned}$$

The pseudo code of searching uniform sharing for **Muls** is shown in Algorithm A1. First, we construct shared functions that satisfies non-completeness and correctness. We partition the quadratic terms and the linear terms over $\mathbb{F}_{2^4}$ in the symbolic representation of $\mathbf{Muls}(x_1+x_2+x_3+x_4)$ into four component functions where each function is independent of at least one share of each input. To decrease the number of partitioning ways, we constrain the number of 4-bit multiplications in each component function to be one. The following equation contains all the possible ways of partitioning that preserves non-completeness, where every coefficient c_{ij} can be 0 or 1:

$$\begin{aligned}
y_1 =&(c_{11}x_2^m+c_{12}x_3^m+c_{13}x_4^m)\times(c_{14}x_2^l+c_{15}x_3^l+c_{16}x_4^l)\\
&+c_{17}\mathrm{SqSc}(x_2^m+x_2^l)+c_{18}\mathrm{SqSc}(x_3^m+x_3^l)+c_{19}\mathrm{SqSc}(x_4^m+x_4^l)\\
y_2 =&(c_{21}x_1^m+c_{22}x_3^m+c_{23}x_4^m)\times(c_{24}x_1^l+c_{25}x_3^l+c_{26}x_4^l)\\
&+c_{27}\mathrm{SqSc}(x_1^m+x_1^l)+c_{28}\mathrm{SqSc}(x_3^m+x_3^l)+c_{29}\mathrm{SqSc}(x_4^m+x_4^l)\\
y_3 =&(c_{31}x_1^m+c_{32}x_2^m+c_{33}x_4^m)\times(c_4x_1^l+c_{35}x_2^l+c_{36}x_4^l)\\
&+c_{37}\mathrm{SqSc}(x_1^m+x_1^l)+c_{38}\mathrm{SqSc}(x_2^m+x_2^l)+c_{39}\mathrm{SqSc}(x_4^m+x_4^l)\\
y_4 =&(c_{41}x_1^m+c_{42}x_2^m+c_{43}x_3^m)\times(c_{44}x_1^l+c_{45}x_2^l+c_{46}x_3^l)\\
&+c_{47}\mathrm{SqSc}(x_1^m+x_1^l)+c_{48}\mathrm{SqSc}(x_2^m+x_2^l)+c_{49}\mathrm{SqSc}(x_3^m+x_3^l).
\end{aligned}\tag{A1}$$

To ensure the correctness, the sum of (y_1,y_2,y_3,y_4) should equal the correct output namely, $(x_1^m+x_2^m+x_3^m+x_4^m)\times(x_1^l+x_2^l+x_3^l+x_4^l)+\mathrm{SqSc}(x_1^m+x_2^m+x_3^m+x_4^m+x_1^l+x_2^l+x_3^l+x_4^l)$. Therefore, the following constraint is imposed on the coefficients of all component functions:

* Corresponding author (email: sunsiwei@is.ac.cn)

$$
\begin{aligned}
c_{21}c_{24} + c_{31}c_{34} + c_{41}c_{44} &= 1\\
c_{31}c_{35} + c_{41}c_{45} &= 1\\
c_{21}c_{25} + c_{41}c_{46} &= 1\\
c_{21}c_{26} + c_{31}c_{36} &= 1\\
c_{32}c_{34} + c_{42}c_{44} &= 1\\
c_{11}c_{14} + c_{32}c_{35} + c_{42}c_{45} &= 1\\
c_{11}c_{15} + c_{42}c_{46} &= 1\\
c_{11}c_{16} + c_{32}c_{36} &= 1\\
c_{22}c_{24} + c_{43}c_{44} &= 1\\
c_{12}c_{14} + c_{43}c_{45} &= 1\\
c_{12}c_{15} + c_{22}c_{25} + c_{43}c_{46} &= 1\\
c_{12}c_{16} + c_{22}c_{26} &= 1\\
c_{23}c_{24} + c_{33}c_{34} &= 1\\
c_{13}c_{14} + c_{33}c_{35} &= 1\\
c_{13}c_{15} + c_{23}c_{25} &= 1\\
c_{13}c_{16} + c_{23}c_{26} + c_{33}c_{36} &= 1\\
c_{17} + c_{38} + c_{48} &= 1\\
c_{18} + c_{28} + c_{49} &= 1\\
c_{19} + c_{29} + c_{39} &= 1\\
c_{27} + c_{37} + c_{47} &= 1.
\end{aligned}
\tag{A2}
$$

Then for the selected sharings, we need to check whether the sharing functions fulfill the uniformity using the property defined in [7]. The pseudo code of checking for uniformity is shown in Algorithm A2. It takes every input value, and iterates all possible input shares to check the distribution of output shares for the uniformity. If the outputs of all component functions are uniform, then the algorithm return true. Otherwise, the algorithm retains false.

Algorithm A1 Algorithm to generate component functions.

Input: Component functions of the sharing: f_1, f_2, f_3, f_4
Output: Component functions which satisfy correctness and non-completeness.

For $(c_{11}, c_{12}, \ldots, c_{19}) = 0$ **to** $2^9 - 1$ **do**
  For $(c_{21}, c_{22}, \ldots, c_{29}) = 0$ **to** $2^9 - 1$ **do**
    For $(c_{31}, c_{32}, \ldots, c_{39}) = 0$ **to** $2^9 - 1$ **do**
      For $(c_{41}, c_{42}, \ldots, c_{49}) = 0$ **to** $2^9 - 1$ **do**
        if $(c_{11}, c_{12}, \ldots, c_{49})$ satisfies the constraint Equation A2 **then**
          //construct the sharing functions (f_1, f_2, f_3, f_4) using Equation A1;
          construct the sharing functions f_1 using $(c_{11}, c_{12}, \ldots, c_{19})$;
          construct the sharing functions f_2 using $(c_{21}, c_{22}, \ldots, c_{29})$;
          construct the sharing functions f_3 using $(c_{31}, c_{32}, \ldots, c_{39})$;
          construct the sharing functions f_4 using $(c_{41}, c_{42}, \ldots, c_{49})$;
          if check the uniformity of the sharing (f_1, f_2, f_3, f_4) **then**
            Print (f_1, f_2, f_3, f_4);
          end if
        end if
      end For
    end For
  end For
end For

Fortunately, we find a uniform sharing which also satisfies non-completeness and correctness as shown in Figure A1 and Equation (A3):

$$
\begin{aligned}
y_1 &= (x_3^m + x_4^m) \times (x_2^l + x_3^l) + \mathrm{SqSc}(x_2^m + x_2^l)\\
y_2 &= (x_1^m + x_3^m) \times (x_1^l + x_4^l) + \mathrm{SqSc}(x_1^m + x_1^l)\\
y_3 &= (x_2^m + x_4^m) \times (x_1^l + x_4^l) + \mathrm{SqSc}(x_4^m + x_4^l)\\
y_4 &= (x_1^m + x_2^m) \times (x_2^l + x_3^l) + \mathrm{SqSc}(x_3^m + x_3^l).
\end{aligned}
\tag{A3}
$$

Besides, the algebraic normal forms of the sharing functions $\mathbf{Muls}_i$ for $1 \leqslant i \leqslant 4$ are presented in the following:

Algorithm A2 Algorithm to check whether a given sharing satisfies uniformity.

Input: Component functions of the sharing: f_1, f_2, f_3, f_4
Output: True, False

For $x = 0$ **to** $2^8 - 1$ **do**
  For $i = 0$ **to** $2^{16} - 1$ **do**
    counter[i]=0;
  end For
  For $x_1 = 0$ **to** $2^8 - 1$ **do**
    For $x_2 = 0$ **to** $2^8 - 1$ **do**
      For $x_3 = 0$ **to** $2^8 - 1$ **do**
        $x_4 = x_1 + x_2 + x_2 + x$
        For $i = 0$ **to** 4 **do**
          $x_i^m \Leftarrow$ the high 4 bits of x_i;
          $x_i^l \Leftarrow$ the low 4 bits of x_i;
        end For
        compute $y_1 = f_1(x_2^m, x_3^m, x_4^m, x_2^l, x_3^l, x_4^l)$
        compute $y_2 = f_2(x_1^m, x_3^m, x_4^m, x_1^l, x_3^l, x_4^l)$
        compute $y_3 = f_3(x_1^m, x_2^m, x_4^m, x_1^l, x_2^l, x_4^l)$
        compute $y_4 = f_4(x_1^m, x_2^m, x_3^m, x_1^l, x_2^l, x_3^l)$
        counter$[y_1 + 2^4 \times y_2 + 2^8 \times y_3 + 2^{12} \times y_4]$++;
      end For
    end For
  end For
  For $i = 0$ **to** $2^{16} - 1$ **do**
    if $counter[i] \neq 2^{12} \vee counter[i] \neq 0$ **then**
      Return False;
      Exit;
    end if
  end For
end For
Return True;

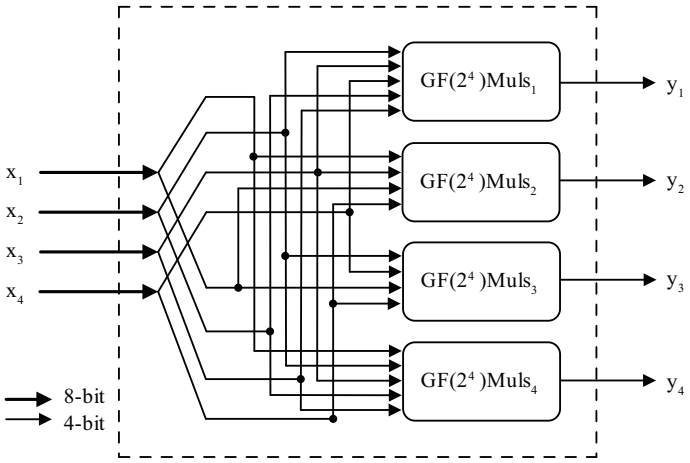

Figure A1 A uniform sharing of the $\mathbb{F}_{2^4}$ multiplier and the $\mathbb{F}_{2^4}$ square scaler as a whole.

$$
\begin{aligned}
(y_1^1,y_1^2,y_1^3,y_1^4) =&\mathbf{Muls}_1(x_2^8,x_2^7,x_2^6,x_2^5,x_3^8,x_3^7,x_3^6,x_3^5,x_4^8,x_4^7,x_4^6,x_4^5,x_2^4,x_2^3,x_2^2,x_2^1,x_3^4,x_3^3,x_3^2,x_3^1)\\
y_1^1 =&(x_3^1+x_4^1)(x_2^5+x_3^5)+(x_3^3+x_4^3)(x_2^5+x_3^5)+(x_3^4+x_4^4)(x_2^5+x_3^5)\\
&+(x_3^2+x_4^2)(x_2^6+x_3^6)+(x_3^3+x_4^3)(x_2^6+x_3^6)+(x_3^1+x_4^1)(x_2^7+x_3^7)\\
&+(x_3^2+x_4^2)(x_2^7+x_3^7)+(x_3^3+x_4^3)(x_2^7+x_3^7)+(x_3^4+x_4^4)(x_2^7+x_3^7)\\
&+(x_3^1+x_4^1)(x_2^8+x_3^8)+(x_3^3+x_4^3)(x_2^8+x_3^8)+x_2^2+x_2^6+x_2^4+x_2^8;\\
y_1^2 =&(x_3^2+x_4^2)(x_2^5+x_3^5)+(x_3^3+x_4^3)(x_2^5+x_3^5)+(x_3^1+x_4^1)(x_2^6+x_3^6)\\
&+(x_3^2+x_4^2)(x_2^6+x_3^6)+(x_3^4+x_4^4)(x_2^6+x_3^6)+(x_3^1+x_4^1)(x_2^7+x_3^7)\\
&+(x_3^3+x_4^3)(x_2^7+x_3^7)+(x_3^2+x_4^2)(x_2^8+x_3^8)+(x_3^4+x_4^4)(x_2^8+x_3^8)+x_2^1+x_2^5\\
&+x_2^3+x_2^7;\\
y_1^3 =&(x_3^1+x_4^1)(x_2^5+x_3^5)+(x_3^2+x_4^2)(x_2^5+x_3^5)+(x_3^3+x_4^3)(x_2^5+x_3^5)\\
&+(x_3^4+x_4^4)(x_2^5+x_3^5)+(x_3^1+x_4^1)(x_2^6+x_3^6)+(x_3^3+x_4^3)(x_2^6+x_3^6)\\
&+(x_3^1+x_4^1)(x_2^7+x_3^7)+(x_3^2+x_4^2)(x_2^7+x_3^7)+(x_3^3+x_4^3)(x_2^7+x_3^7)\\
&+(x_3^1+x_4^1)(x_2^8+x_3^8)+(x_3^4+x_4^4)(x_2^8+x_3^8)+x_2^3+x_2^7+x_2^4+x_2^8;\\
y_1^4 =&(x_3^1+x_4^1)(x_2^5+x_3^5)+(x_3^3+x_4^3)(x_2^5+x_3^5)+(x_3^2+x_4^2)(x_2^6+x_3^6)\\
&+(x_3^4+x_4^4)(x_2^6+x_3^6)+(x_3^1+x_4^1)(x_2^7+x_3^7)+(x_3^4+x_4^4)(x_2^7+x_3^7)\\
&+(x_3^2+x_4^2)(x_2^8+x_3^8)+(x_3^3+x_4^3)(x_2^8+x_3^8)+(x_3^4+x_4^4)(x_2^8+x_3^8)+x_2^4+x_2^8;
\end{aligned}
$$

$$
\begin{aligned}
(y_2^1,y_2^2,y_2^3,y_2^4) =&\mathbf{Muls}_2(x_1^8,x_1^7,x_1^6,x_1^5,x_3^8,x_3^7,x_3^6,x_3^5,x_1^4,x_1^3,x_1^2,x_1^1,x_4^4,x_4^3,x_4^2,x_4^1)\\
y_2^1 =&(x_1^1+x_3^1)(x_1^5+x_4^5)+(x_1^3+x_3^3)(x_1^5+x_4^5)+(x_1^4+x_3^4)(x_1^5+x_4^5)\\
&+(x_1^2+x_3^2)(x_1^6+x_4^6)+(x_1^3+x_3^3)(x_1^6+x_4^6)+(x_1^1+x_3^1)(x_1^7+x_4^7)\\
&+(x_1^2+x_3^2)(x_1^7+x_4^7)+(x_1^3+x_3^3)(x_1^7+x_4^7)+(x_1^4+x_3^4)(x_1^7+x_4^7)\\
&+(x_1^1+x_3^1)(x_1^8+x_4^8)+(x_1^3+x_3^3)(x_1^8+x_4^8)+x_1^2+x_1^6+x_1^4+x_1^8;\\
y_2^2 =&(x_1^2+x_3^2)(x_1^5+x_4^5)+(x_1^3+x_3^3)(x_1^5+x_4^5)+(x_1^1+x_3^1)(x_1^6+x_4^6)\\
&+(x_1^2+x_3^2)(x_1^6+x_4^6)+(x_1^4+x_3^4)(x_1^6+x_4^6)+(x_1^1+x_3^1)(x_1^7+x_4^7)\\
&+(x_1^3+x_3^3)(x_1^7+x_4^7)+(x_1^2+x_3^2)(x_1^8+x_4^8)+(x_1^4+x_3^4)(x_1^8+x_4^8)+x_1^1+x_1^5\\
&+x_1^3+x_1^7;\\
y_2^3 =&(x_1^1+x_3^1)(x_1^5+x_4^5)+(x_1^2+x_3^2)(x_1^5+x_4^5)+(x_1^3+x_3^3)(x_1^5+x_4^5)\\
&+(x_1^4+x_3^4)(x_1^5+x_4^5)+(x_1^1+x_3^1)(x_1^6+x_4^6)+(x_1^3+x_3^3)(x_1^6+x_4^6)\\
&+(x_1^1+x_3^1)(x_1^7+x_4^7)+(x_1^2+x_3^2)(x_1^7+x_4^7)+(x_1^3+x_3^3)(x_1^7+x_4^7)\\
&+(x_1^1+x_3^1)(x_1^8+x_4^8)+(x_1^4+x_3^4)(x_1^8+x_4^8)+x_1^3+x_1^7+x_1^4+x_1^8;\\
y_2^4 =&(x_1^1+x_3^1)(x_1^5+x_4^5)+(x_1^3+x_3^3)(x_1^5+x_4^5)+(x_1^2+x_3^2)(x_1^6+x_4^6)\\
&+(x_1^4+x_3^4)(x_1^6+x_4^6)+(x_1^1+x_3^1)(x_1^7+x_4^7)+(x_1^4+x_3^4)(x_1^7+x_4^7)\\
&+(x_1^2+x_3^2)(x_1^8+x_4^8)+(x_1^3+x_3^3)(x_1^8+x_4^8)+(x_1^4+x_3^4)(x_1^8+x_4^8)+x_1^4+x_1^8;
\end{aligned}
$$

$$
\begin{aligned}
(y_3^1,y_3^2,y_3^3,y_3^4) =&\mathbf{Muls}_3(x_2^8,x_2^7,x_2^6,x_2^5,x_4^8,x_4^7,x_4^6,x_4^5,x_4^4,x_1^3,x_1^2,x_1^1,x_4^4,x_4^3,x_4^2,x_4^1)\\
y_3^1 =&(x_2^1+x_4^1)(x_1^5+x_4^5)+(x_2^3+x_4^3)(x_1^5+x_4^5)+(x_2^4+x_4^4)(x_1^5+x_4^5)\\
&+(x_2^2+x_4^2)(x_1^6+x_4^6)+(x_2^3+x_4^3)(x_1^6+x_4^6)+(x_2^1+x_4^1)(x_1^7+x_4^7)\\
&+(x_2^2+x_4^2)(x_1^7+x_4^7)+(x_2^3+x_4^3)(x_1^7+x_4^7)+(x_2^4+x_4^4)(x_1^7+x_4^7)\\
&+(x_2^1+x_4^1)(x_1^8+x_4^8)+(x_2^3+x_4^3)(x_1^8+x_4^8)+x_4^2+x_4^6+x_4^4+x_4^8;\\
y_3^2 =&(x_2^2+x_4^2)(x_1^5+x_4^5)+(x_2^3+x_4^3)(x_1^5+x_4^5)+(x_2^1+x_4^1)(x_1^6+x_4^6)\\
&+(x_2^2+x_4^2)(x_1^6+x_4^6)+(x_2^4+x_4^4)(x_1^6+x_4^6)+(x_2^1+x_4^1)(x_1^7+x_4^7)\\
&+(x_2^3+x_4^3)(x_1^7+x_4^7)+(x_2^2+x_4^2)(x_1^8+x_4^8)+(x_2^4+x_4^4)(x_1^8+x_4^8)+x_4^1+x_4^5\\
&+x_4^3+x_4^7;\\
y_3^3 =&(x_2^1+x_4^1)(x_1^5+x_4^5)+(x_2^2+x_4^2)(x_1^5+x_4^5)+(x_2^3+x_4^3)(x_1^5+x_4^5)\\
&+(x_2^4+x_4^4)(x_1^5+x_4^5)+(x_2^1+x_4^1)(x_1^6+x_4^6)+(x_2^3+x_4^3)(x_1^6+x_4^6)\\
&+(x_2^1+x_4^1)(x_1^7+x_4^7)+(x_2^2+x_4^2)(x_1^7+x_4^7)+(x_2^3+x_4^3)(x_1^7+x_4^7)\\
&+(x_2^1+x_4^1)(x_1^8+x_4^8)+(x_2^4+x_4^4)(x_1^8+x_4^8)+x_4^3+x_4^7+x_4^4+x_4^8;\\
y_3^4 =&(x_2^1+x_4^1)(x_1^5+x_4^5)+(x_2^3+x_4^3)(x_1^5+x_4^5)+(x_2^2+x_4^2)(x_1^6+x_4^6)\\
&+(x_2^4+x_4^4)(x_1^6+x_4^6)+(x_2^1+x_4^1)(x_1^7+x_4^7)+(x_2^4+x_4^4)(x_1^7+x_4^7)\\
&+(x_2^2+x_4^2)(x_1^8+x_4^8)+(x_2^3+x_4^3)(x_1^8+x_4^8)+(x_2^4+x_4^4)(x_1^8+x_4^8)+x_4^4+x_4^8;
\end{aligned}
$$

$$
\begin{aligned}
(y_4^1, y_4^2, y_4^3, y_4^4) =& \mathbf{Muls}_4(x_1^8, x_1^7, x_1^6, x_1^5, x_2^8, x_2^7, x_2^6, x_2^5, x_3^8, x_3^7, x_3^6, x_3^5, x_2^4, x_2^3, x_2^2, x_2^1, x_3^4, x_3^3, x_3^2, x_3^1)\\
y_4^1 =& (x_1^1 + x_2^1)(x_2^5 + x_3^5) + (x_1^3 + x_2^3)(x_2^5 + x_3^5) + (x_1^4 + x_2^4)(x_2^5 + x_3^5)\\
&+ (x_1^2 + x_2^2)(x_2^6 + x_3^6) + (x_1^3 + x_2^3)(x_2^6 + x_3^6) + (x_1^1 + x_2^1)(x_2^7 + x_3^7)\\
&+ (x_1^2 + x_2^2)(x_2^7 + x_3^7) + (x_1^3 + x_2^3)(x_2^7 + x_3^7) + (x_1^4 + x_2^4)(x_2^7 + x_3^7)\\
&+ (x_1^1 + x_2^1)(x_2^8 + x_3^8) + (x_1^3 + x_2^3)(x_2^8 + x_3^8) + x_3^2 + x_3^6 + x_3^4 + x_3^8;\\
y_4^2 =& (x_1^2 + x_2^2)(x_2^5 + x_3^5) + (x_1^3 + x_2^3)(x_2^5 + x_3^5) + (x_1^1 + x_2^1)(x_2^6 + x_3^6)\\
&+ (x_1^2 + x_2^2)(x_2^6 + x_3^6) + (x_1^4 + x_2^4)(x_2^6 + x_3^6) + (x_1^1 + x_2^1)(x_2^7 + x_3^7)\\
&+ (x_1^3 + x_2^3)(x_2^7 + x_3^7) + (x_1^2 + x_2^2)(x_2^8 + x_3^8) + (x_1^4 + x_2^4)(x_2^8 + x_3^8) + x_3^1 + x_3^5\\
&+ x_3^3 + x_3^7;\\
y_4^3 =& (x_1^1 + x_2^1)(x_2^5 + x_3^5) + (x_1^2 + x_2^2)(x_2^5 + x_3^5) + (x_1^3 + x_2^3)(x_2^5 + x_3^5)\\
&+ (x_1^4 + x_2^4)(x_2^5 + x_3^5) + (x_1^1 + x_2^1)(x_2^6 + x_3^6) + (x_1^3 + x_2^3)(x_2^6 + x_3^6)\\
&+ (x_1^1 + x_2^1)(x_2^7 + x_3^7) + (x_1^2 + x_2^2)(x_2^7 + x_3^7) + (x_1^3 + x_2^3)(x_2^7 + x_3^7)\\
&+ (x_1^1 + x_2^1)(x_2^8 + x_3^8) + (x_1^4 + x_2^4)(x_2^8 + x_3^8) + x_3^3 + x_3^7 + x_3^4 + x_3^8;\\
y_4^4 =& (x_1^1 + x_2^1)(x_2^5 + x_3^5) + (x_1^3 + x_2^3)(x_2^5 + x_3^5) + (x_1^2 + x_2^2)(x_2^6 + x_3^6)\\
&+ (x_1^4 + x_2^4)(x_2^6 + x_3^6) + (x_1^1 + x_2^1)(x_2^7 + x_3^7) + (x_1^4 + x_2^4)(x_2^7 + x_3^7)\\
&+ (x_1^2 + x_2^2)(x_2^8 + x_3^8) + (x_1^3 + x_2^3)(x_2^8 + x_3^8) + (x_1^4 + x_2^4)(x_2^8 + x_3^8) + x_3^4 + x_3^8.
\end{aligned}
$$

Appendix B Searching for a uniform sharing of the $\mathbb{F}_{2^4}$ inverter

Previous TIs of the AES S-box based on the tower field architecture employ mainly two types of TI for the $\mathbb{F}_{2^4}$ inverter. The first type [7] uses four input shares and four output shares, requires one clock cycle, and is not uniform. The second type [2] uses five input shares and five output shares, requires one clock cycle, and is uniform.

We propose a new TI for the $\mathbb{F}_{2^4}$ inverter, where the input and output are divided into 4 shares and 2 shares respectively. Our implementation, depicted in Figure B1, is implemented in two stages.

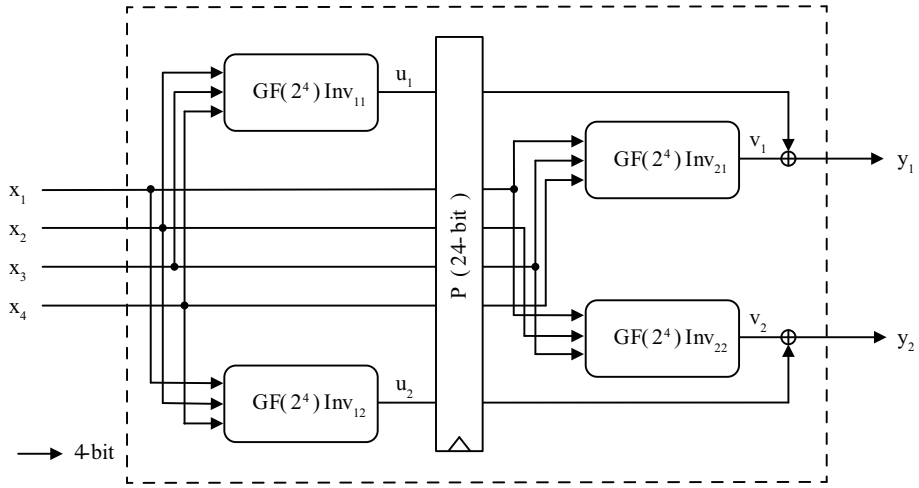

Figure B1 A uniform sharing for the $\mathbb{F}_{2^4}$ inverter.

The $\mathbb{F}_{2^4}$ inverter can be represented as a vectorial Boolean function as follows:

$$
\begin{aligned}
(y^1, y^2, y^3, y^4) &= Inv(x^1, x^2, x^3, x^4)\\
y^1 &= x^3 + x^4 + x^3x^1 + x^3x^2 + x^4x^3x^2\\
y^2 &= x^4 + x^3x^1 + x^3x^2 + x^4x^2 + x^4x^3x^2\\
y^3 &= x^1 + x^2 + x^3x^1 + x^4x^1 + x^4x^2x^1\\
y^4 &= x^2 + x^3x^1 + x^4x^1 + x^4x^2 + x^4x^3x^2.
\end{aligned}
$$

The subcomponents $\mathbf{Inv}_{ij} : \mathbb{F}_{2^4} \times \mathbb{F}_{2^4} \times \mathbb{F}_{2^4} \to \mathbb{F}_{2^4}$ for $i, j \in \{1, 2\}$ are vectorial Boolean functions, whose algebraic normal forms are presented in the following:

$$
\begin{aligned}
(u_1^1,u_1^2,u_1^3,u_1^4) =&\mathbf{Inv}_{11}(x_2^1,x_2^2,x_2^3,x_2^4,x_3^1,x_3^2,x_3^3,x_3^4,x_4^1,x_4^2,x_4^3,x_4^4)\\
u_1^1 =&(x_2^2+x_3^2+x_4^2)(x_2^3+x_3^3+x_4^3)(x_2^4+x_3^4+x_4^4)+(x_2^2+x_3^2+x_4^2)(x_2^3+x_3^3+x_4^3)\\
&+x_2^4+(x_2^1+x_3^1+x_4^1)(x_2^3+x_3^3+x_4^3)+x_2^3;\\
u_1^2 =&(x_2^1+x_3^1+x_4^1)(x_2^3+x_3^3+x_4^3)(x_2^4+x_3^4+x_4^4)+(x_2^1+x_3^1+x_4^1)(x_2^3+x_3^3+x_4^3)\\
&+x_2^4+(x_2^2+x_3^2+x_4^2)(x_2^4+x_3^4+x_4^4)+(x_2^2+x_3^2+x_4^2)(x_2^3+x_3^3+x_4^3);\\
u_1^3 =&(x_2^1+x_3^1+x_4^1)(x_2^4+x_3^4+x_4^4)(x_2^2+x_3^2+x_4^2)+(x_2^1+x_3^1+x_4^1)(x_2^4+x_3^4+x_4^4)\\
&+x_2^2+(x_2^1+x_3^1+x_4^1)(x_2^3+x_3^3+x_4^3)+x_2^1;\\
u_1^4 =&(x_2^1+x_3^1+x_4^1)(x_2^3+x_3^3+x_4^3)(x_2^2+x_3^2+x_4^2)+(x_2^1+x_3^1+x_4^1)(x_2^3+x_3^3+x_4^3)\\
&+x_2^2+(x_2^1+x_3^1+x_4^1)(x_2^4+x_3^4+x_4^4)+(x_2^2+x_3^2+x_4^2)(x_2^4+x_3^4+x_4^4);
\end{aligned}
$$

$$
\begin{aligned}
(u_2^1,u_2^2,u_2^3,u_2^4) =&\mathbf{Inv}_{12}(x_1^1,x_1^2,x_1^3,x_1^4,x_2^1,x_2^2,x_2^3,x_2^4,x_4^1,x_4^2,x_4^3,x_4^4)\\
u_2^1 =&x_1^2x_1^3x_2^4+x_1^2x_2^3x_1^4+x_2^2x_1^3x_1^4+x_1^2x_2^3x_2^4+x_2^2x_1^3x_2^4+x_2^2x_2^3x_1^4+x_1^2x_2^3x_4^4+x_2^2x_1^3x_4^4\\
&+x_1^2x_4^3x_2^4+x_2^2x_4^3x_1^4+x_4^2x_1^3x_2^4+x_4^2x_2^3x_1^4+x_1^2x_2^3+x_1^3x_2^2+x_4^4+x_1^1x_2^3+x_1^3x_2^1\\
&+x_4^3;\\
u_2^2 =&x_1^1x_1^3x_2^4+x_1^1x_2^3x_1^4+x_2^1x_1^3x_1^4+x_1^1x_2^3x_2^4+x_2^1x_1^3x_2^4+x_2^1x_2^3x_1^4+x_1^1x_2^3x_4^4+x_2^1x_1^3x_4^4\\
&+x_1^1x_4^3x_2^4+x_2^1x_4^3x_1^4+x_4^1x_1^3x_2^4+x_4^1x_2^3x_1^4+x_1^1x_2^3+x_1^3x_2^1+x_4^4+x_1^2x_2^4+x_1^4x_2^2\\
&+x_1^2x_2^3+x_1^3x_2^2;\\
u_2^3 =&x_1^1x_1^4x_2^2+x_1^1x_2^4x_1^2+x_2^1x_1^4x_1^2+x_1^1x_2^4x_2^2+x_2^1x_1^4x_2^2+x_2^1x_2^4x_1^2+x_1^1x_2^4x_4^2+x_2^1x_1^4x_4^2\\
&+x_1^1x_4^4x_2^2+x_2^1x_4^4x_1^2+x_4^1x_1^4x_2^2+x_4^1x_2^4x_1^2+x_1^1x_2^4+x_1^4x_2^1+x_4^2+x_1^1x_2^3+x_1^3x_2^1\\
&+x_4^1;\\
u_2^4 =&x_1^1x_1^3x_2^2+x_1^1x_2^3x_1^2+x_2^1x_1^3x_1^2+x_1^1x_2^3x_2^2+x_2^1x_1^3x_2^2+x_2^1x_2^3x_1^2+x_1^1x_2^3x_4^2+x_2^1x_1^3x_4^2\\
&+x_1^1x_4^3x_2^2+x_2^1x_4^3x_1^2+x_4^1x_1^3x_2^2+x_4^1x_2^3x_1^2+x_1^1x_2^3+x_1^3x_2^1+x_4^2+x_1^1x_2^4+x_1^4x_2^1\\
&+x_1^2x_2^4+x_1^4x_2^2;
\end{aligned}
$$

$$
\begin{aligned}
(v_1^1,v_1^2,v_1^3,v_1^4) =&\mathbf{Inv}_{21}(x_1^1,x_1^2,x_1^3,x_1^4,x_3^1,x_3^2,x_3^3,x_3^4,x_4^1,x_4^2,x_4^3,x_4^4)\\
v_1^1 =&x_1^2(x_3^3+x_3^3)(x_3^4+x_4^4)+x_1^3(x_3^2+x_4^2)(x_3^4+x_4^4)+x_1^4(x_3^2+x_4^2)(x_3^3+x_4^3)\\
&+x_1^2x_1^3(x_3^4+x_4^4)+x_1^2x_1^4(x_3^3+x_4^3)+x_1^3x_1^4(x_3^2+x_4^2)+x_1^2x_1^3x_1^4+x_1^2(x_3^3+x_4^3)\\
&+x_1^3(x_3^2+x_4^2)+x_1^2x_1^3+x_3^4+x_1^1(x_3^3+x_4^3)+x_1^3(x_3^1+x_4^1)+x_1^1x_1^3+x_3^3;\\
v_1^2 =&x_1^1(x_3^3+x_3^3)(x_3^4+x_4^4)+x_1^3(x_3^1+x_4^1)(x_3^4+x_4^4)+x_1^4(x_3^1+x_4^1)(x_3^3+x_4^3)\\
&+x_1^1x_1^3(x_3^4+x_4^4)+x_1^1x_1^4(x_3^3+x_4^3)+x_1^3x_1^4(x_3^1+x_4^1)+x_1^1x_1^3x_1^4+x_1^1(x_3^3+x_4^3)\\
&+x_1^3(x_3^1+x_4^1)+x_1^1x_1^3+x_3^4+x_1^2(x_3^4+x_4^4)+x_1^4(x_3^2+x_4^2)+x_1^2x_1^4\\
&+x_1^2(x_3^3+x_4^3)+x_1^3(x_3^2+x_4^2)+(x_1^2x_1^3);\\
v_1^3 =&x_1^1(x_3^4+x_4^4)(x_3^2+x_4^2)+x_1^4(x_3^1+x_4^1)(x_3^2+x_4^2)+x_1^2(x_3^1+x_4^1)(x_3^4+x_4^4)\\
&+x_1^1x_1^4(x_3^2+x_4^2)+x_1^1x_1^2(x_3^4+x_4^4)+x_1^4x_1^2(x_3^1+x_4^1)+x_1^1x_1^4x_1^2+x_1^1(x_3^4+x_4^4)\\
&+x_1^4(x_3^1+x_4^1)+x_1^1x_1^4+x_3^2+x_1^1(x_3^3+x_4^3)+x_1^3(x_3^1+x_4^1)+x_1^1x_1^3+x_3^1;\\
v_1^4 =&x_1^1(x_3^3+x_3^3)(x_3^2+x_4^2)+x_1^3(x_3^1+x_4^1)(x_3^2+x_4^2)+x_1^2(x_3^1+x_4^1)(x_3^3+x_4^3)\\
&+x_1^1x_1^3(x_3^2+x_4^2)+x_1^1x_1^2(x_3^3+x_4^3)+x_1^3x_1^2(x_3^1+x_4^1)+x_1^1x_1^3x_1^2+x_1^1(x_3^3+x_4^3)\\
&+x_1^3(x_3^1+x_4^1)+x_1^1x_1^3+x_3^2+x_1^1(x_3^4+x_4^4)+x_1^4(x_3^1+x_4^1)+x_1^1x_1^4\\
&+x_1^2(x_3^4+x_4^4)+x_1^4(x_3^2+x_4^2)+x_1^2x_1^4;
\end{aligned}
$$

$$
\begin{aligned}
(v_2^1,v_2^2,v_2^3,v_2^4) =&\mathbf{Inv}_{22}(x_1^1,x_1^2,x_1^3,x_1^4,x_2^1,x_2^2,x_2^3,x_2^4,x_3^1,x_3^2,x_3^3,x_3^4)\\
v_2^1 =&x_1^2x_2^3x_3^4+x_1^2x_3^3x_2^4+x_2^2x_1^3x_3^4+x_2^2x_3^3x_1^4+x_3^2x_1^3x_2^4+x_3^2x_2^3x_1^4+x_1^4+x_1^3;\\
v_2^2 =&x_1^1x_2^3x_3^4+x_1^1x_3^3x_2^4+x_2^1x_1^3x_3^4+x_2^1x_3^3x_1^4+x_3^1x_1^3x_2^4+x_3^1x_2^3x_1^4+x_1^4;\\
v_2^3 =&x_1^1x_2^4x_3^2+x_1^1x_3^4x_2^2+x_2^1x_1^4x_3^2+x_2^1x_3^4x_1^2+x_3^1x_1^4x_2^2+x_3^1x_2^4x_1^2+x_1^2+x_1^1;\\
v_2^4 =&x_1^1x_2^3x_3^2+x_1^1x_3^3x_2^2+x_2^1x_1^3x_3^2+x_2^1x_3^3x_1^2+x_3^1x_1^3x_2^2+x_3^1x_2^3x_1^2+x_1^2.
\end{aligned}
$$

Appendix C Hardware architecture

The overall architecture is shown in Figure C1. The detail of the two 128-bit register arrays for the encryption state are presented in Figure C2. According to the state arrays, the output bytes of the S-box are stored into state array along the path of $33 \to 23 \to 13 \to 03 \to 32 \to \cdots 00$. For the implementation of ShiftRows, we employ the technique proposed

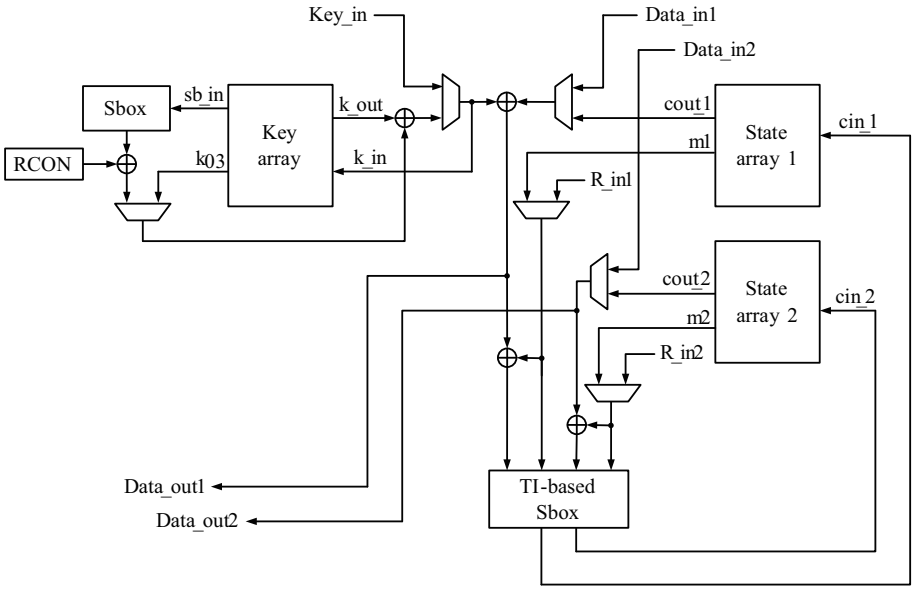

Figure C1 Hardware architecture of the byte-serial AES.

by [4] to reduce the latency, where the ShiftRows is performed in the 20th cycle of every round while the state is loading the output bytes of the S-box. Then, the MixCols is performed in the last four cycles, and the outputs are stored into the rightmost column (registers marked with 03, 13, 23, 33) and shifted horizontally.

The state for key schedule algorithm is depicted in Figure C3. Except using an unshared S-box, the implementation of the key schedule largely follows [4] with necessary tweaks to make it in tune with the encryption process.

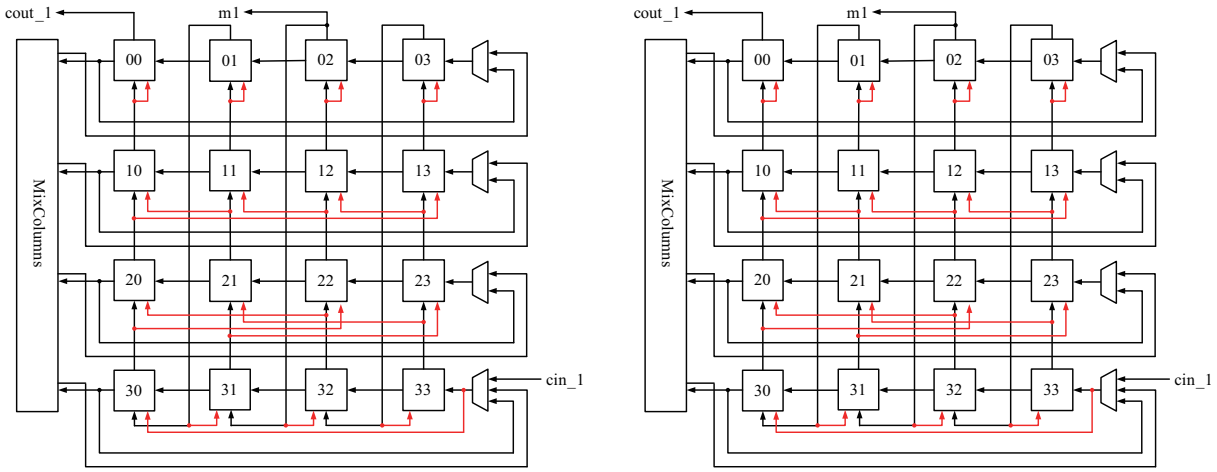

Figure C2 Architecture of two state arrays for AES TI.

Appendix D Synthesis results

We synthesized our design using Synopsys Design Compiler 2014.09-SP3 with the `UMC` 180`nm` standard cell library, and the results are summarized in Table D1. The results show that our realization costs 5850 GE and 256 clock cycles totally, which is the first TI of AES which consumes less than 6000 `GE` without fresh randomness. The table also shows implementation results for the subcomponents of our AES TI, where our shared AES S-box costs 2034 GE. Besides, compared with the previous implementation that demands no fresh randomness [5], our implementation significantly reduces the number of clock cycles required for one encryption from 2804 to 256 .

1) without *compile_ultra* flag
2) including the MixCols

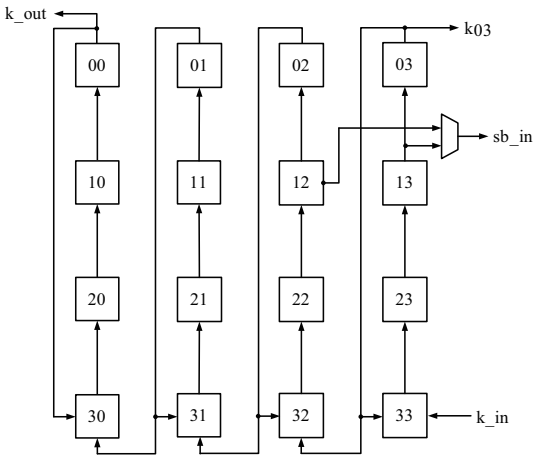

Figure C3 Architecture of the key array for AES TI.

Table D1 Implementation results and comparisons with existing AES TIs

Design	Library	Area[GE] *compile* / *_ultra*	Randomness [bits]	Clock Cycles
Related Implementations				
Moradi et al. [6]	UMC 180nm	11114 / 11031	48	266
Bilgin et al. [7]	UMC 180nm	8119 / 7282	32	246
Cnudde et al. [8]	NanGate 45nm	6681 / 6340	54	276
Ueno et al. [9]	TSMC 65nm	6321 / 6053	64	219
Gross et al. [10]	UMC 90nm	6000[1)]	18	246
Wegener et al. [11]	UMC 180nm	7600[1)]	0	2804
Our Implementation				
AES	UMC 180nm	6195 / 5850	0	256
Shared S-box		2133 / 2034		
Key array		770 / 712		
State arrays [2)]		2703 / 2552		
Unshared S-box		234 / 225		
Control		355 / 327		

Appendix E Leakage Analysis

We collect 10 million power traces for TI realization and TI realization with PRNG respectively. Example power traces are shown in Figure E1. We evaluate the effectiveness of our designs by applying the Test Vector Leakage Assessment (TVLA) based on Welch's t-test (*a.k.a* non-specific t-test [12, 13]), and the results obtained with 10 million traces are shown in Figure E2 and Figure E3 for TI realization and TI realization with PRNG.

As is commonly believed, the absolute t-value of less than 4.5 indicates a high confidence in the resistance against standard first-order attacks. Therefore, we conclude that our AES TI is secure against standard first-order attacks under the condition of 10 million traces. On the other hand, as shown in Figure E3, our TI exhibits obvious leakage with second-order analysis.

References

1 Bilgin B, Gierlichs B, Nikova S, et al. Trade-offs for threshold implementations illustrated on AES. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2015, 34(7): 1188–1200

2 Bilgin B, Gierlichs B, Nikova S, et al. A more efficient AES threshold implementation. In: Proceedings of International Conference on Cryptology in Africa, Marrakesh, Morocco, 2014. 267–284

3 Chen C, Farmani M, Eisenbarth T. A tale of two shares: why two-share threshold implementation seems worthwhile-and why it is not. In: Proceedings of International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, 2016. 819–843

4 Gross H, Mangard S, Korak T. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. In: Proceedings of the ACM Workshop on Theory of Implementation Security, Vienna, Austria, 2016. 3

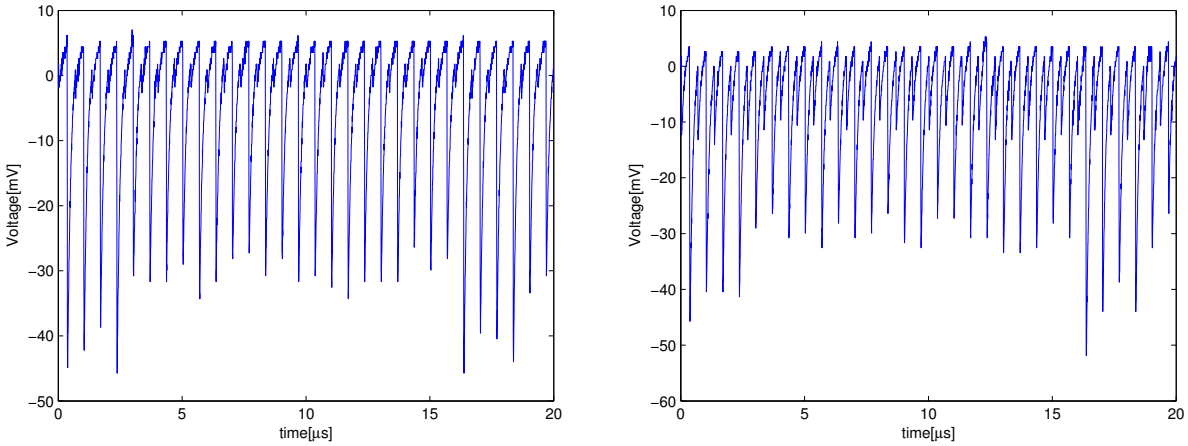

Figure E1 Power trace of the AES TI without PRNG (left) and with PRNG (right)

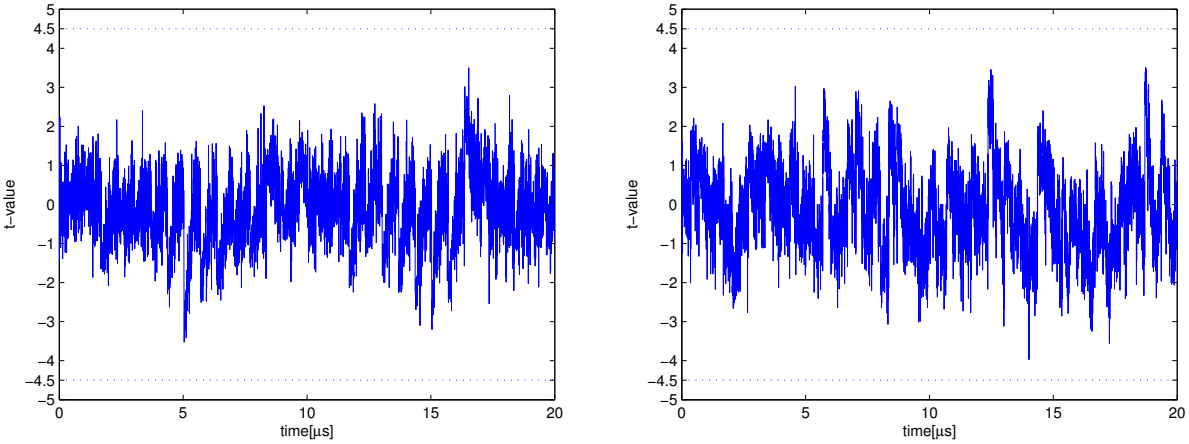

Figure E2 First-order t-test results without PRNG (left) and with PRNG (right)

5 Wegener F, Moradi A. A first-order SCA resistant AES without fresh randomness. In: Proceedings of International Workshop on Constructive Side-Channel Analysis and Secure Design, Singapore, 2018. 245–262

6 Moradi A, Poschmann A, Ling S, et al. Pushing the limits: a very compact and a threshold implementation of AES. In: Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, 2011. 69–88

7 Bilgin B, Gierlichs B, Nikova S, et al. Trade-offs for threshold implementations illustrated on AES. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2015, 34(7): 1188–1200

8 De Cnudde T, Reparaz O, Bilgin B, et al. Masking AES with $d+1$ Shares in Hardware. In: Proceedings of International Conference on Cryptographic Hardware and Embedded Systems, Santa Barbara, CA, USA, 2016. 194–212

9 Ueno R, Homma N, Aoki T. Toward more efficient DPA-resistant AES hardware architecture based on threshold implementation. In: Proceedings of International Workshop on Constructive Side-Channel Analysis and Secure Design, Paris, France, 2017. 50–64

10 Groß H, Mangard S, Korak T. An efficient side-channel protected AES implementation with arbitrary protection order. In: Proceedings of Cryptographers Track at the RSA Conference, San Francisco, CA, USA, 2017. 95–112

11 Wegener F, Moradi A. A first-order SCA resistant AES without fresh randomness. In: Proceedings of International Workshop on Constructive Side-Channel Analysis and Secure Design, Singapore, 2018. 245–262

12 Schneider T, Moradi A. Leakage assessment methodology. In: Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems, Saint-Malo, France, 2015. 495–513

13 Goodwill G, Jun B, Jaffe J, et al. A testing methodology for side-channel resistance validation. In: Proceedings of NIST Non-Invasive Attack Testing Workshop, Nara, Japan, 2011. 115–136

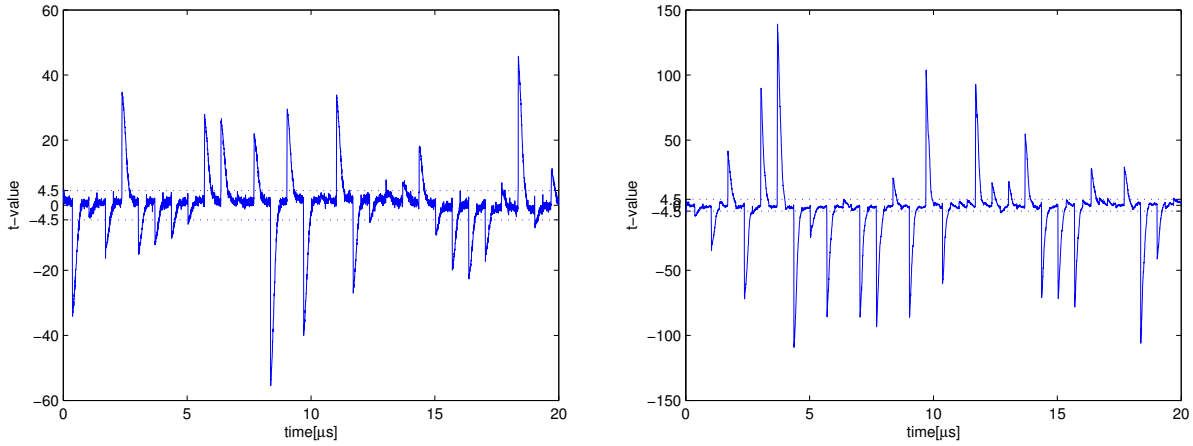

Figure E3 Second-order t-test results without PRNG (left) and with PRNG (right)