A Fast and Provably Secure Higher-Order Masking of AES S-box

2011.09.29

HeeSeok Kim, Seokhie Hong, and Jongin Lim
Center for Information Security Technologies

Korea University CIST Center for Information Security Technologies

Outline

- History of Higher-Order Masking of AES
- Higher-Order Differential Power Analysis & Higher-Order Masking
- Advanced Encryption Standard (AES) S-box & Inversion over a Composite Field
- A Fast Higher-Order Masking of AES S-box
- Performance Analysis & Implementation Results

History of Higher-Order Masking of AES

- Higher-Order masking schemes : **Countermeasures to provide the perfect security against higher-order DPA (d^{th}-order masking scheme can block d^{th}-order DPA)**
- In 2006, Kai Schramm and Christof Paar proposed the first higher-order masking of AES.
  - They did not prove a security of their masking method : In 2007, it has been broken for the order of 3 or more.
  - It requires much computation time.
- In 2008, provably secure 2^{nd}-order masking method was proposed.
  - It is dedicated to order 2 and also requires much computation time.
- In 2010, provably secure higher-order masking method was proposed.
  - The security for all order was proven.
  - This method can considerably reduce the computation time.
  - But, it is still slow and not practical to use in embedded processors.

Higher-Order DPA & Higher-Order Masking of AES

- **Differential Power Analysis**
  - A statistical power analysis of many executions of the same algorithm
  - The power consumption is strongly related to the internal state of the device.

- **Masking methods**
  - Algorithmic techniques : inexpensive and secure against a 1^{st}-order DPA
  - A random mask is added to every sensitive variable.
  - Instantaneous power leakage is independent of sensitive variables : 1^{st}-order DPA attack feasible

- **2^{nd}-order differential power analysis against masking methods**
  - X : sensitive variable, M : random mask
  - $X \oplus M$ (Masked sensitive variable) : processed at t_0
  - M : processed at t_1
  - $P(t_0)$: power consumption at t_0, $P(t_1)$: power consumption at t_1
  - Correlation between the product of two power signals and hypothetical power $f(X, K_h)$

$$\rho([P(t_0) - E(P(t_0))][P(t_1) - E(P(t_1))], f(X, K_h))$$

Higher-Order DPA & Higher-Order Masking of AES

- d^{th}-order masking methods
  - randomly split X into $(d+1)$-tuple $(X_0, X_1, X_2, \ldots, X_d)$ s.t. $X_0 \oplus X_1 \oplus X_2 \oplus \ldots \oplus X_d = X$
  - X : sensitive variable, $M_1, M_2, \ldots., M_d$: d random masks
  - $X \oplus M_1 \oplus M_2 \oplus M_3 \oplus \ldots \oplus M_{d-1} \oplus M_d$ (Masked sensitive variable) : processed at t_0
  - M_i's : processed at t_i

- $(d+1)^{th}$-order differential power analysis against d^{th}-order masking methods
  - $P(t_i)$: power consumption at t_i
  - Correlation between the product of $(d+1)$ power signals and hypothetical power $f(X, K_h)$

$$\rho\left(\prod_{i=0}^{d}[P(t_i) - E(P(t_i))], f(X, K_h)\right)$$

- d^{th}-order masking methods
  - randomly split every sensitive variable X of an original cipher into $(d+1)$-tuple $(X_0, X_1, X_2, \ldots, X_d)$ s.t. $\perp_{i=0}^{d} X_i = X$ where $\perp$ is any group operation.

	Original cipher	Masked cipher
Encryption algorithm	$c \leftarrow e(m, k)$	$(c_0, c_1, \ldots, c_d) \leftarrow e'((m_0, m_1, \ldots, m_d), (k_0, k_1, \ldots, k_d))$ $(c = \perp_{i=0}^{d} c_i,\ m = \perp_{i=0}^{d} m_i,\ k = \perp_{i=0}^{d} k_i)$
Intermediate value	I	$(I_0, I_1, \ldots, I_d)$ s.t. $I = \perp_{i=0}^{d} I_i$
Linear operation	$O \leftarrow L(I)$	$(O_0, O_1, \ldots, O_d) \leftarrow L'((I_0, I_1, \ldots, I_d))$ where $O_i = L(I_i)$ ➔ If $\perp = \oplus$, $O = \perp_{i=0}^{d} O_i = L(\perp_{i=0}^{d} I_i) = L(I)$
Non-linear operation	$O \leftarrow NL(I)$	??

Higher-Order DPA & Higher-Order Masking of AES

- **Higher-order masking scheme of non-linear operation**
  - **Most of the cost for higher-order masking scheme is required by non-linear operation.**
  - **In the case of AES, to construct the higher-order masking scheme in all previous works, the most important consideration has been to mask S-box operation.**

- **Higher-order masking of AES S-box [18]**
  - **AES S-box is defined by a multiplicative inverse $x^{(-1)}$ and an affine transformation A_f**
  - **Masking the affine transformation $O \leftarrow A_f(I)$ is easy**
    - If d is even, the d^{th}-order masking of A_f is $(O_0, O_1, \ldots, O_d) \leftarrow A_f'((I_0, I_1, \ldots, I_d))$ where $O_i = A_f(I_i)$
    - If d is odd, the d^{th}-order masking of A_f is $(O_0, O_1, \ldots, O_d) \leftarrow A_f'((I_0, I_1, \ldots, I_d))$ where $O_0 = A_f(I_0) \oplus 0x63$ and $O_i = A_f(I_i)$ $(i \neq 0)$
    - The d^{th}-order masking of $x^{(-1)}$ is constructed by the **d^{th}-order secure exponentiation**.

Higher-Order DPA & Higher-Order Masking of AES

- d^{th}-order secure exponentiation [18] : constructed by d^{th}-order secure square and multiplication
  - d^{th}-order secure square : t squaring is linear operation over $\mathbb{F}_{256}$

$$X^{2^t} = \oplus_{i=0}^{d} X_i^{2^t}$$

  - d^{th}-order secure multiplication : non-linear operation, difficulty to mask
  - $(c_0, c_1, \dots, c_d)$ = SecMult$((a_0, a_1, \dots, a_d), (b_0, b_1, \dots, b_d))$ s.t. $c = \oplus_{i=0}^{d} c_i = \oplus_{i=0}^{d} a_i \oplus_{i=0}^{d} b_i = ab$
  - SecMult function requires $(d+1)^2$ $GF(2^8)$ multiplications
  - The addition chain of x^{254} to minimize the number of multiplications :

$$x \xrightarrow[S]{} x^2 \xrightarrow[M]{} x^3 \xrightarrow[2S]{} x^{12} \xrightarrow[M]{} x^{15} \xrightarrow[4S]{} x^{240} \xrightarrow[M]{} x^{252} \xrightarrow[M]{} x^{254}$$

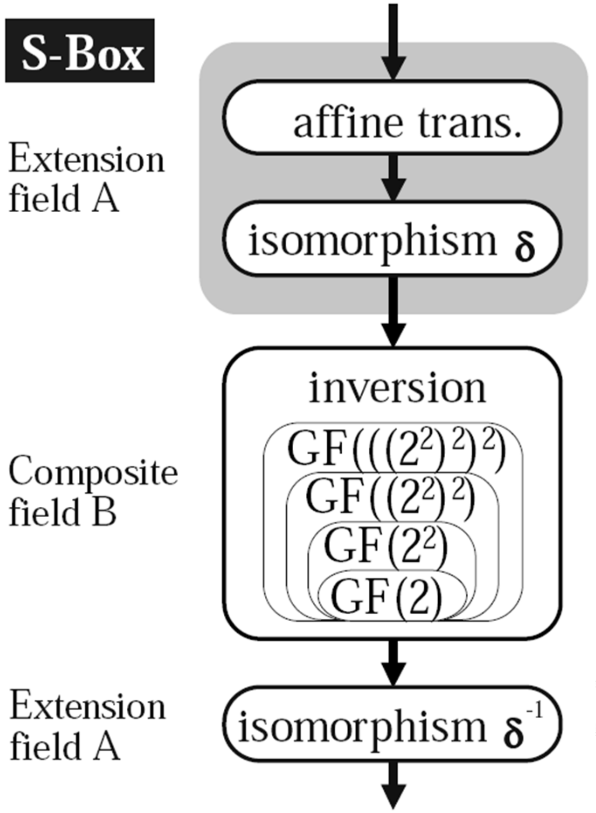
  - The requirement of $4(d+1)^2$ $GF(2^8)$ multiplications : <u>$12(d+1)^2$ table lookup operations</u> (log/alog tables)

SubBytes of AES & Inversion for SubBytes

- **SubBytes of AES**
  - $S : GF(2^8) \rightarrow GF(2^8)$
  - $S(x) = Mx^{(-1)} \oplus v$ **where** M **is an 8x8** $GF(2)$**-matrix, and** v **is an 8x1** $GF(2)$**-vector.**
  - $x^{(-1)} = x^{-1}$ **in** $GF(2^8)$ **(except if** $x = 0$ **then** $x^{(-1)} = 0$**)**

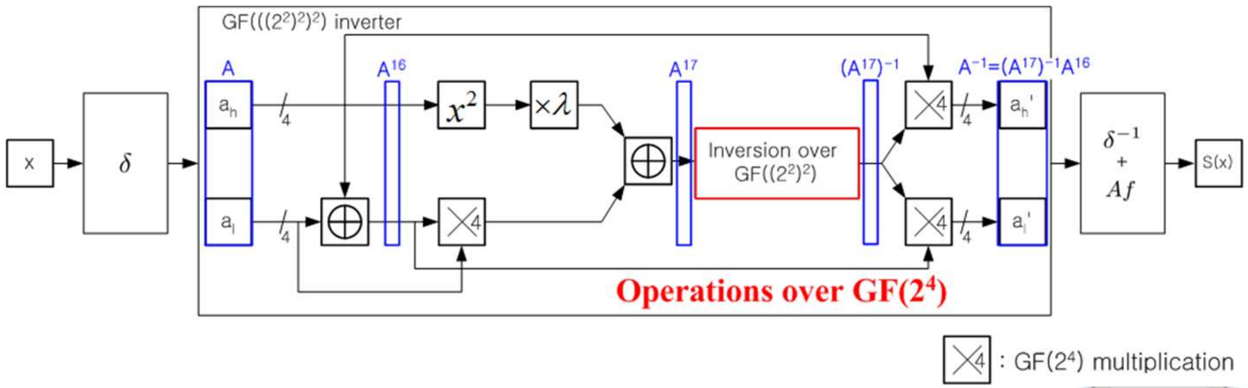
- **Inversion Operation over a Composite Field [21]**
  - **This operation has been proposed to reduce the cost of AES SubBytes.**
  - **Order of Operations**
    - Transform an element over $GF(2^8)$ into an element over the composite field having low inversion cost.
    - Compute the inverse of this transformed element over composite field.
    - Carry out the inverse mapping into the element over $GF(2^8)$.

- **Inversion Operation over a Composite Field [21]**



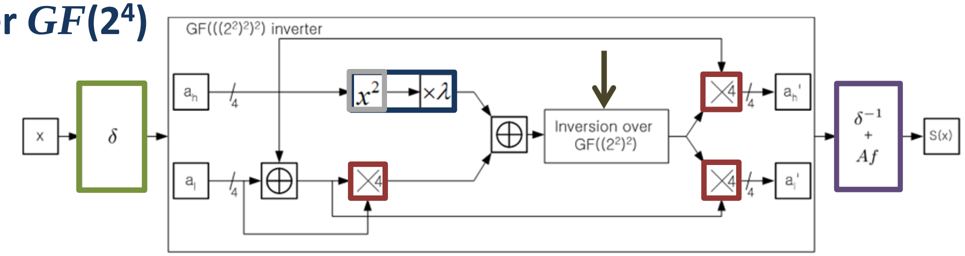
$$GF(2^2) \quad : \quad P_0(x) = x^2 + x + 1, \text{ where } P_0(\alpha) = 0,$$
$$GF((2^2)^2) \quad : \quad P_1(x) = x^2 + x + \alpha, \text{ where } P_1(\beta) = 0,$$
$$GF(((2^2)^2)^2) \quad : \quad P_2(x) = x^2 + x + \lambda, \text{ where } \lambda = (\alpha + 1)\beta, \ P_2(\gamma) = 0.$$

Inversion operation over the composite field	• **STEP 1 :** $a_h\gamma + a_l = \delta(x)$ • **STEP 2 :** $d = \lambda a_h^2 + a_l(a_h + a_l)$ • **STEP 3 :** $d' = d^{-1}$ • **STEP 4 :** $(a_h', a_l') = (d' a_h, d'(a_h + a_l))$ • **STEP 5 :** $\delta^{-1}(a_h'\gamma + a_l')$



A Fast Higher-Order Masking of AES S-box

- **Main purpose :**
  - **Now, it is not practical to use higher-order masking schemes in the embedded processors because of their speed.**
  - **Reduce running time of the higher-order masking scheme**
- **Idea : use the inversion operation over the composite field and precomputation tables**
- **6 precomputation tables (total requirement for 816 bytes of ROM)**
  - **Squaring table $T1$ over $GF(2^4)$**
  - **Two squaring table $T2$ over $GF(2^4)$**
  - **Squaring-scalar multiplication table $T3$ over $GF(2^4)$**
  - **Multiplication table $T4$ over $GF(2^4)$**
  - **Isomorphism table $T5$**
  - **Inverse isomorphism-Affine transformation table $T6$**



A Fast Higher-Order Masking of AES S-box

Algorithm. d^{th}-order masking of AES S-box

Input : $(x_0, x_1, \ldots, x_d)$ s.t. $x = \bigoplus_{i=0}^{d} x_i$

Output : $(y_0, y_1, \ldots, y_d)$ s.t. y = S-box(x)= $\bigoplus_{i=0}^{d} y_i$

1(a). $(H_0||L_0, H_1||L_1, \ldots, H_d||L_d) = (T5[x_0], T5[x_1], \ldots, T5[x_d])$

1(b). $(w_0, w_1, \ldots, w_d) = (T3[H_0], T3[H_1], \ldots, T3[H_d])$

1(c). $(t_0, t_1, \ldots, t_d) = (H_0 \oplus L_0, H_1 \oplus L_1, \ldots, H_d \oplus L_d)$

2. $(L_0, L_1, \ldots, L_d)$ = SecMult4($(t_0, t_1, \ldots, t_d), (L_0, L_1, \ldots, L_d)$)

3. $(w_0, w_1, \ldots, w_d) = (w_0 \oplus L_0, w_1 \oplus L_1, \ldots, w_d \oplus L_d)$

4. $(w_0, w_1, \ldots, w_d)$ = SecInv($(w_0, w_1, \ldots, w_d)$)

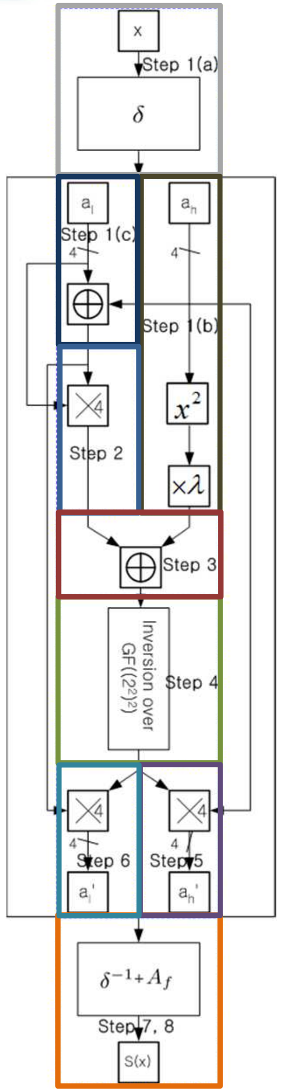
5. $(H_0, H_1, \ldots, H_d)$ = SecMult4($(w_0, w_1, \ldots, w_d), (H_0, H_1, \ldots, H_d)$)

6. $(L_0, L_1, \ldots, L_d)$ = SecMult4($(w_0, w_1, \ldots, w_d), (L_0, L_1, \ldots, L_d)$)

7. $(y_0, y_1, \ldots, y_d) = (T6[H_0||L_0], T6[H_1||L_1], \ldots, T6[H_d||L_d])$

8. If d is odd, $y_0 = y_0 \oplus 0x63$

9. Return $(y_0, y_1, \ldots, y_d)$



A Fast Higher-Order Masking of AES S-box

- **Masking non-linear operations**
  - **Masking $GF(2^4)$ inversion (SecInv function)**
    - Using the composite field operation over $GF((2^2)^2)$ similarly to the masked operation over $GF(((2^2)^2)^2)$: requires as many table lookup operations as that over $GF(((2^2)^2)^2)$.
    - The addition chain of x^{14} to minimize the number of multiplications :

$$x \underset{S}{\longrightarrow} x^2 \underset{M}{\longrightarrow} x^3 \underset{2S}{\longrightarrow} x^{12} \underset{M}{\longrightarrow} x^{14}$$

Algorithm. $GF(2^4)$ SecInv function

Input : $(x_0, x_1, \ldots, x_d)$ s.t. $x = \bigoplus_{i=0}^{d} x_i$

Output : $(y_0, y_1, \ldots, y_d)$ s.t. $y = x^{14} = \bigoplus_{i=0}^{d} y_i$

1. $(w_0, w_1, \ldots, w_d) = (T1[x_0], T1[x_1], \ldots, T1[x_d])$ // x^2
2. RefreshMasks($(w_0, w_1, \ldots, w_d)$) // Eliminate the dependence between two input tuples
3. $(z_0, z_1, \ldots, z_d)$ = SecMult4($(w_0, w_1, \ldots, w_d)$, $(x_0, x_1, \ldots, x_d)$) // x^3
4. $(z_0, z_1, \ldots, z_d) = (T2[z_0], T2[z_1], \ldots, T2[z_d])$ // x^{12}
5. $(y_0, y_1, \ldots, y_d)$ = SecMult4($(z_0, z_1, \ldots, z_d)$, $(w_0, w_1, \ldots, w_d)$) // x^{14}

A Fast Higher-Order Masking of AES S-box

- **Masking non-linear operations**
  - **Masking $GF(2^4)$ multiplication (SecMult4 function)**
    - Using the idea of [18]
    - $(d+1)^2$ $GF(2^4)$ multiplications : $(d+1)^2$ table lookup operations by $T4$ table
    - Our higher-order masking of AES S-box needs 5 SecMult4 function calls : $5(d+1)^2$ table lookup operations

Performance Analysis

Algorithm. d^{th}-order masking of AES S-box

Input : $(x_0, x_1, \ldots, x_d)$ s.t. $x = \oplus_{i=0}^{d} x_i$

Output : $(y_0, y_1, \ldots, y_d)$ s.t. y = S-box(x)= $\oplus_{i=0}^{d} y_i$

$1_{(a)}$. $(H_0||L_0, H_1||L_1, \ldots, H_d||L_d) = (T5[x_0], T5[x_1], \ldots, T5[x_d])$

$1_{(b)}$. $(w_0, w_1, \ldots, w_d) = (T3[H_0], T3[H_1], \ldots, T3[H_d])$

$1_{(c)}$. $(t_0, t_1, \ldots, t_d) = (H_0 \oplus L_0, H_1 \oplus L_1, \ldots, H_d \oplus L_d)$

2. $(L_0, L_1, \ldots, L_d)$ = SecMult4($(t_0, t_1, \ldots, t_d)$, $(L_0, L_1, \ldots, L_d)$)

3. $(w_0, w_1, \ldots, w_d) = (w_0 \oplus L_0, w_1 \oplus L_1, \ldots, w_d \oplus L_d)$

4. $(w_0, w_1, \ldots, w_d)$ = SecInv($(w_0, w_1, \ldots, w_d)$)

5. $(H_0, H_1, \ldots, H_d)$ = SecMult4($(w_0, w_1, \ldots, w_d)$, $(H_0, H_1, \ldots, H_d)$)

6. $(L_0, L_1, \ldots, L_d)$ = SecMult4($(w_0, w_1, \ldots, w_d)$, $(L_0, L_1, \ldots, L_d)$)

7. $(y_0, y_1, \ldots, y_d) = (T6[H_0||L_0], T6[H_1||L_1], \ldots, T6[H_d||L_d])$

8. If d is odd, $y_0 = y_0 \oplus 0x63$

9. Return $(y_0, y_1, \ldots, y_d)$

- **4-bit shift operation may require 4 instruction calls unless the single instruction carrying out 4-bit shift is supported.**
- **However, some microcontrollers like 8051 and AVR family support a single SWAP operation, which swaps high and low nibbles in a register.**
- **To get the random nibbles, we split 1 random byte into two nibbles.**

Performance Analysis

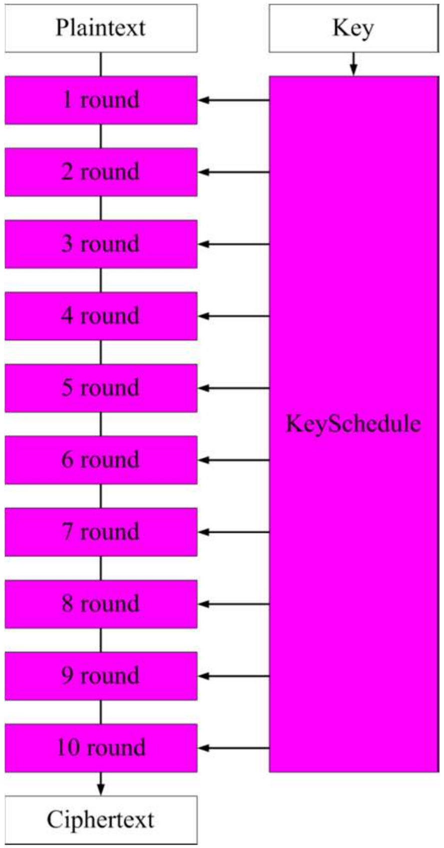
Table 1. Comparison of two d-th order masked S-box schemes in terms of the total number of operations

	Ours	[18]
Table Lookup	$5d^2 + 13d + 8$	$12d^2 + 31d + 19$
XOR	$10d^2 + 16d + 5$	$8d^2 + 12d$
Random Bits	$10d^2 + 14d$	$16d^2 + 32d$
etc	4-bit logical shift : $\frac{5}{4}d^2 + \frac{15}{4}d + 2$,	8-bit Addition : $8(d+1)^2$,
	8-bit bitwise AND : $\frac{5}{4}d^2 + \frac{15}{4}d + 2$	8-bit logical AND : $4(d+1)^2$

– **Implementation of [18]**

- Using log/alog tables
- Remove the reduction operation modulo 255 : to improve the computation speed
- Remove the conditional branch : to eliminate the possibility of SPA

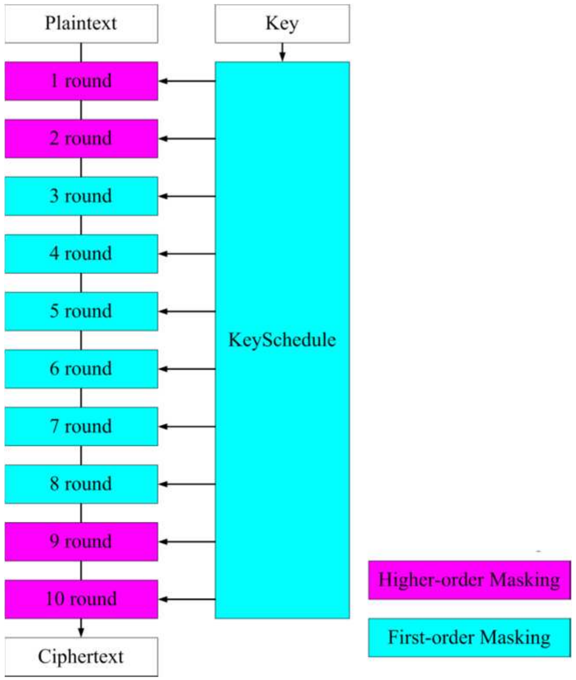
Full-round Higher-Order Masking



Method	Cycles (x 10^3 cc)			Table Size (Bytes)	
	KeyExpand	Encryption	Total	RAM	ROM
Original AES					
No Masking (Straightforward AES)	2.2	9.0	11.2	0	256
First Order Masking					
ACNS'06 [9] (No dummy, No Shuffling)	4.6	14.9	19.5	256	256
Second Order Masking					
FSE'08 [17] (Complete second-order masking)	247.4	950.0	1197.4	256	256
CHES'10 [18] (Complete second-order masking)	144.1	531.2	675.4	0	768
Ours (Complete second-order masking)	66.2	199.3	265.5	0	816
Third Order Masking					
CHES'10 [18] (Complete third-order masking)	293.4	1102.9	1396.3	0	768
Ours (Complete third-order masking)	114.6	346.8	461.3	0	816

- **AES-128 in C-language for ATmega128 8-bit architecture**
- **2.54 (second) and 3.03 (third) faster than [18]**

Reduced Masking



Method	Cycles (x 10^3 cc)			Table Size (Bytes)	
	KeyExpand	Encryption	Total	RAM	ROM
Original AES					
No Masking (Straightforward AES)	2.2	9.0	11.2	0	256
First Order Masking					
ACNS'06 [9] (No dummy, No Shuffling)	4.6	14.9	19.5	256	256
Second Order Masking					
Ours (KeyExpand (first) Enc. (1,2,9,10:second, 3~8:first))	5.2	90.6	95.8	256	1062
Third Order Masking					
Ours (KeyExpand (first) Enc. (1,2,9,10:third, 3~8:first))	5.5	149.6	155.1	256	1062

- **Reduced Masking : <u>higher-order masking on 1,2,9,10 rounds</u>, <u>first-order masking on KeyExpand and the rest of the rounds</u> : higher-order DPA generally attacks the first and last few rounds**
- **First-order masking on KeyExpand and the rest of the rounds : the security against the analysis such as [8] and [12]**
- **just 8.6 (second) and 13.8 (third) slower than the straightforward AES**

Thank you.