Improved side channel attack on the block cipher NOEKEON

Changyong Peng [a] Chuangying Zhu [b] Yuefei Zhu [a] Fei Kang [a]

[a] *Zhengzhou Information Science and Technology Institute,Zhengzhou, China,*
Email: pengchangyong@tom.com,39463021@qq.com,zyf0136@sina.com,k_fei@sina.com

[b] *School of Computer and Control, Guillin University of Electronic Technology, Guilin, China*

Abstract

NOEKEON is a block cipher having key-size 128 and block size 128,proposed by Daemen, J et al.Shekh Faisal Abdul-Latip et al. give a side channel attack(under the single bit leakage model) on the cipher at ISPEC 2010.Their analysis shows that one can recover the 128-bit key of the cipher, by considering a one-bit information leakage from the internal state after the second round, with time complexity of $O(2^{68})$ evaluations of the cipher, and data complexity of about 2^{10} chosen plaintexts.Our side channel attack improves upon the previous work of Shekh Faisal Abdul-Latip et al. from two aspects. First, we use the Hamming weight leakage model(Suppose the Hamming weight of the lower 64 bits and the higher 64 bits of the output of the first round can be obtained without error) which is a more relaxed leakage assumption, supported by many previously known practical results on side channel attacks, compared to the more challenging leakage assumption that the adversary has access to the "exact" value of the internal state bits as used by Shekh Faisal Abdul-Latip et al. Second, our attack has also a reduced complexity compared to that of Shekh Faisal Abdul-Latip et al. Namely, our attack of recovering the 128-bit key of NOEKEON has a time complexity 20.1 seconds on a PC with 2.6 GHZ CPU and 8G RAM and data complexity of 99 known plaintexts; whereas, that of Shekh Faisal Abdul-Latip et al. has time complexity of $O(2^{68})$ and needs about 2^{10} chosen plaintexts.

Keywords: NOEKEON, symbolic computation, Gröbner Basis, side channel attack, algebraic-side channel attack, method of formal coding-side channel attack

1. Introduction

Classical cryptanalysis generally considers adversaries getting black box access to the cryptographic primitives they target, e.g. the inputs and outputs of a block cipher. However,considering the practical implementations of a block cipher, especially in resource limited systems such as smart cards, there is a stronger attack model, namely the side channel attack model, where the adversary is given more power by having access to some *limited* information leaked about the internal state of the cipher. This information leakage can be via physical side channels, such as timing, electrical power consumption, electromagnetic radiation, probing, etc.

One of a recent trend is to combine the classical cryptanalysis such as algebraic cryptanalysis with side channel attacks, e.g. the work of Bogdanov, Kizhvatov and Pyshkin at INDOCRYPT 2008 [1], the side channel cube attack introduced by Dinur and Shamir in [2] which was used to attack the block cipher PRESENT[5,6,7], NOEKEON [9],and the ASCA(Algebraic-Side Channel Attack) proposed by Renauld et al. in [3,4].

In this paper,we propose the Method of Formal Coding-Side Channel Attack(MFCSCA) by combing the method of formal coding and side channel attack.The idea of MFC[10] is to find the algebraic expression of each bit in the ciphertext as an XOR sum of products of the bits of the plaintext and the master key.The XOR-sum form of the ciphertext is also known as the algebraic normal form(ANF).The formal manipulations of these expressions may decrease the key search effort.Schaumuller-Bichl [11,12] studied this method and concluded that it requires an enormous amount of computer memory which makes the whole approach impractical.However,by combing with side channel attack,especially when the leaked information comes from earlier rounds,MFC is practical.The idea of MFCSCA is to give the explicit algebraic ex-

pressions of the leaked information,with the plaintext bits and master key bits as variables.These algebraic expressions can be obtained by modern symbolic computation software such as Mathematica[16].If the algebraic expressions are not very complex,the corresponding equation system can be solved by Gröbner basis-based method [15] or using a SAT solver as in [14].We selected Gröbner basis-based method in this paper.

As an application, we exhibit an MFCSCA attack against the block cipher NOEKEON.

Comparing Side Channel Cube attack of [9] with Our attack. The leakage model used by Shekh Faisal Abdul-Latip et al. [9] assumes that adversary has access to the exact value of some of the internal state bits after each round. We note that obtaining the exact value of the internal state bits in practice will require a probe station that allows the attacker to monitor the value of a specific bit position in the internal state during the encryption or decryption process. This implies an intrusive physical measurement and is known to involve a wide range of difficulties such as penetrating the device to access its internals and guessing which bit position is being recorded. To relax the leakage model, in contrast, we assume the Hamming weight leakage as a more common side channel leakage model, e.g. see [18-22].In this paper,we suppose that the Hamming weight of the lower 64 bits and the higher 64 bits of the output of the first round can be obtained without error. In this paper we do not consider these specific issues and countermeasures about the actual physical aspects of the implementation attacks and how information leakage can be measured. Rather, we assume the Hamming weight leakage side channel model as an abstract attack model, and concentrate on investigating the security of ciphers against MFCSCA attacks in this attack model.

Our side channel attack improves upon the previous work of Shekh Faisal Abdul-Latip et al. from two aspects. First, we use the Hamming weight leakage model which is a more relaxed leakage assumption, supported by many previously known practical results on side channel attacks, compared to the more challenging leakage assumption that the adversary has access to the "exact" value of the internal state bits as used by Shekh Faisal Abdul-Latip et al. Second, our attack has also a reduced complexity compared to that of Shekh Faisal Abdul-Latip et al. Namely, our attack of recovering the 128-bit key of NOEKEON has a time complexity 10 seconds on a PC with 2.6 GHZ CPU and 8G RAM and data complexity of 99 known plaintexts; whereas, that of Shekh Faisal Abdul-Latip et al. has time complexity of $O(2^{68})$ and needs about 2^{10} chosen plaintexts.

2. Description of the Method of Formal Coding-Side Channel Attack(MFCSCA)

MFCSCA can also be thought of an extended version of algebraic-side channel[3,4].Algebraic side-channel attacks are made of three separate steps.

(1)Offline phase 1: algebraic description of the cryptosystem.

(2)Online measurement phase to obtain leaked information.

(3)Offine phase 2: equation system solving by SAT method.

MFCSCA has also the above 3 steps.However,the step 1 and 2 of MFCSCA is different from that of ASCA.

In step (1) of ASCA,the algebraic description of a block cipher consists of two parts:one is the equation system following the idea of algebraic attack[13].This is an **implicit** equation system by setting the intermediate state as variables.The other is the equation system generated from the leaked Hamming weight.In MFCSCA,we first give the **direct explicit** representation of the outputs of earlier rounds of a block cipher in terms of the bits of plaintext and master key.This is just the idea of the method of formal coding.This can be done with the help of symbolic computation software such as Mathematica [16].Then the Hamming weight(mod 2) of the outputs can be explicitly represented by the bits of plaintext and master key.

In step (3) of ASCA,SAT solving technique [14] is used to find the solution of the equation system.In step (3) of MFCSCA,we choose Gröbner basis-based [15] methods to find the solution.

3. Experimental results–Method of formal coding-side channel attack on NOEKEON

3.1. Description of NOEKEON

NOEKEON is an iterated 128-bit block cipher with 128-bit keys, which runs in 16 rounds. Each round consists of some subfunctions, a linear function called Theta, three rotations called Pi1, 32 parallel 4-bit S-box(see Table 1) lookups called Gamma and three rotations called Pi2. All functions take a 128-bit text input. The function Theta takes as input also the 128-bit round key. To avoid round symmetries a round constant is added to 32 bits of the ciphertexts in each round. After the 16 rounds, an output transformation is applied. This consists of the addition of a round constant and a final application of Theta. The output transformation and the nature of the individual round function components allow for very similar encryption

Table 1: Specification of the S-box in the function Gamma.

i	0	1	2	3	4	5	6	7
S[i]	7	A	2	C	4	8	F	0
i	8	9	A	B	C	D	E	F
S[i]	5	9	1	E	3	D	B	6

and decryption routines. Figure 1 illustrates the round function in NOEKEON, where C_i is a round dependent constant and $K_0, ..., K_3$ are the 32-bit subkeys.The least significant byte of C_0 to C_{16} are(in hexidecimal):80,1b,36,6c,d8,ab,4d,9a,2f,5e,bc,63,c6,97,35,6a,d4. (We refer to [8] for a complete description of the cipher).

Let $A_r = a_0^r a_1^r a_2^r a_3^r$ denote the 128-bit internal state after round r; where a_i^r's are 32-bit words, and A_0 contains the input plaintext P to the cipher.Then NOEKEON encryption algorithm can be described as follows:

For $r = 0; r < 16; r++$
  $A_{r+1} = Pi2(Gamma(Pi1(Theta(A_r, K))))$;
$A_{17} = Theta(A_{16}, K)$

The specification of NOEKEON [8], provides a key schedule which converts the 128-bit Master Key (i.e. the original key) into a 128-bit Working Key, which is used in the round function. However, the use of the key schedule is optional. If related-key attack scenarios [23] are not of a concern, then the key schedule is not applied (i.e. the Master Key is used directly as the Working Key), and the cipher is called to be used in the direct- key mode. Otherwise, it operates in the indirect-key mode, where the Master Key is processed by the key schedule algorithm to produce the Working Key.

3.2. Algebraic expressions of the bits of the output of the 1st round of NOEKEON

In this section,we give the algebraic expressions of the bits of the output of the 1st round of NOEKEON with the plaintext bits and master key bits as variables.

Notation:
$K = k_0k_1...k_{127}$: master key of NOEKEON,
$P = p_0p_1...p_{127}$: plaintext of NOEKEON,
$B = b_0b_1...b_{127}$: output of the 1st round of NOEKEON.
Note:In this paper we number bits from zero with bit zero(the most significant bit) on the left of a block or word.

```
Theta[key_, a_] := Module[
   {K0, K1, K2, K3, a0, a1, a2, a3, temp},
   a0 = a[[1 ;; 32]]; a1 = a[[33 ;; 64]];
   a2 = a[[65 ;; 96]]; a3 = a[[97 ;; 128]];
   K0 = key[[1 ;; 32]]; K1 = key[[33 ;; 64]];
   K2 = key[[65 ;; 96]]; K3 = key[[97 ;; 128]];
   temp = a0 + a2;
   temp = temp + RotateLeft[temp, 8]
     + RotateRight[temp, 8];
   a1 = a1 + temp; a3 = a3 + temp;
   a0 = a0 + K0; a1 = a1 + K1; a2 = a2 + K2;
   a3 = a3 + K3; temp = a1 + a3;
   temp = temp + RotateLeft[temp, 8]
     + RotateRight[temp, 8];
   a0 = a0 + temp; a2 = a2 + temp;
   Return[{a0, a1, a2, a3} // Flatten]];
```

Figure 2: Mathematica code of the subfunction Theta of NOEKEON

We use the powerful symbolic computation software Mathematica [16] to obtain the result.To obtain the algebraic expressions of each bit of $B = b_0b_1...b_{127}$, output of the 1st round of NOEKEON,we need only give the symbolic encryption code for NOEKEON.This can easily be done with Mathematica in just the same way we write a C encryption code for NOEKEON.First,we give the Mathematica code of each of the subfunctions(Theta,Pi1,Pi2 and Gamma,see figure 1).The Mathematica code of each of the subfunctions are very simple.We give only the source code of Theta here(see figure 2).

The whole symbolic encryption code(in direct-key mode) is given in appendix A.We test the correctness of the symbolic encryption code using the test vector given in [24](see Table 2).

we use the following Mathematica code to test the correctness of the symbolic encryption code(see appendix A).

```
m = Table[0, {128}];
k = Table[0, {128}];
c = encryption[m, k];
BaseForm[FromDigits[c, 2], 16]
m = Table[1, {128}];
k = Table[1, {128}];
c = encryption[m, k];
BaseForm[FromDigits[c, 2], 16]
k = IntegerDigits[16^^b1656851699e29fa24b70148503d2dfc,
2, 128];
m = IntegerDigits[16^^2a78421b87c7d0924f26113f1d1349b2,
```

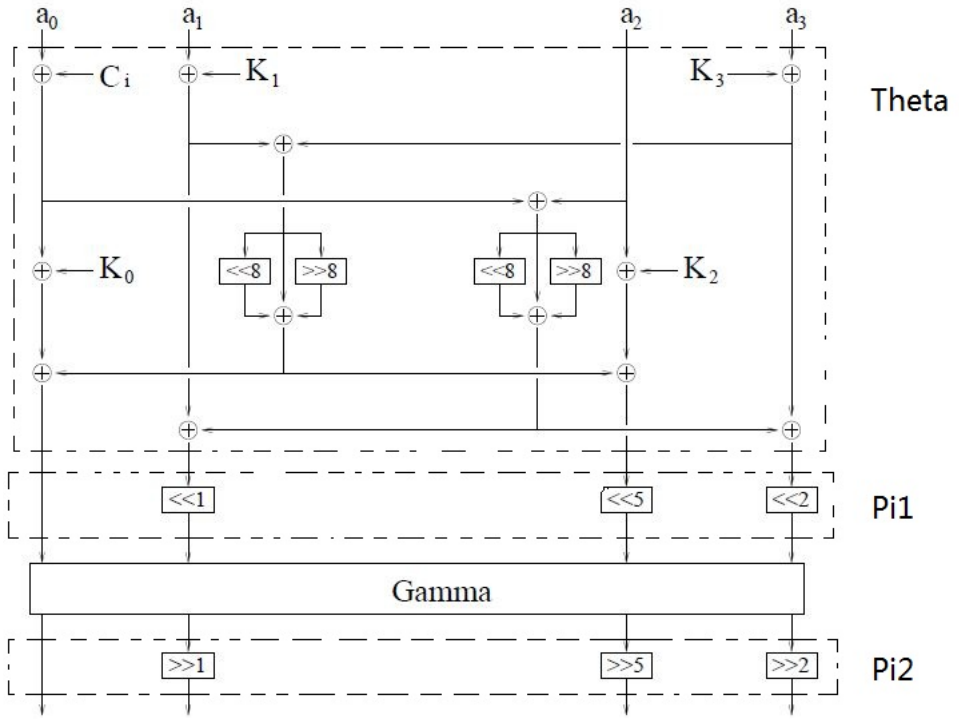

Figure 1: Round function of NOEKEON

2, 128];
c = encryption[m, k];
BaseForm[FromDigits[c, 2], 16]

Obtaining the algebraic expressions of the output of the 1st round:

Let $K = k_0k_1...k_{127}$(pure symbol) be the master key and $P = p_0p_1...p_{127}$(pure symbol) be the plaintext of NOEKEON.Then we run the symbolic encryption of NOEKEON and output the state $B = b_0b_1...b_{127}$ after the 1st round.

The followings(see appendix A for details) are the Mathematica code for obtaining the algebraic expressions of each bit of $B = b_0b_1...b_{127}$.

K = Table[ToExpression[StringJoin["k",ToString[i]]], {i,0,127}];
P = Table[ToExpression[StringJoin["p",ToString[i]]], {i,0,127}];
c = encryption[P,K,1]

Here, encryption[P,K,1] means the output $B = b_0b_1...b_{127}$ of round 1. In this way can obtain the algebraic expressions of each bit of $B = b_0b_1...b_{127}$.

Table 2: Correctness test result of our symbolic encryption code(in direct-key mode) for NOEKEON.

key1	00000000000000000000000000000000
plaintext1	00000000000000000000000000000000
ciphertext1	b1656851699e29fa24b70148503d2dfc
our ciphertext1	b1656851699e29fa24b70148503d2dfc
key2	ffffffffffffffffffffffffffffffff
plaintext2	ffffffffffffffffffffffffffffffff
ciphertext2	2a78421b87c7d0924f26113f1d1349b2
our ciphertext2	2a78421b87c7d0924f26113f1d1349b2
key3	b1656851699e29fa24b70148503d2dfc
plaintext3	2a78421b87c7d0924f26113f1d1349b2
ciphertext3	e2f687e07b75660ffc372233bc47532c
our ciphertext3	e2f687e07b75660ffc372233bc47532c

We list only three of them,b_0, b_{63}, b_{127}:

$b_0 = k_{98}+p_2+p_{10}+p_{26}+p_{66}+p_{74}+p_{90}+p_{98}+\Big(k_0+k_{32}+k_{33}+k_{37}+k_{40}+k_{45}+k_{56}+k_{61}+k_{69}+k_{96}+k_{98}+k_{101}+k_{104}+k_{109}+k_{120}+k_{125}+p_0+p_1+p_2+p_9+p_{10}+p_{25}+p_{26}+p_{32}+p_{33}+p_{37}+p_{40}+p_{45}+p_{56}+p_{61}+p_{65}+p_{66}+p_{69}+p_{73}+p_{74}+p_{89}+p_{90}+p_{96}+p_{98}+p_{101}+p_{104}+p_{109}+p_{120}+p_{125}+\Big(k_{98}+p_2+p_{10}+p_{26}+p_{66}+p_{74}+p_{90}+p_{98}+1\Big)\Big(k_{37}+k_{45}+k_{61}+k_{69}+k_{101}+k_{109}+k_{125}+p_{37}+p_{45}+p_{61}+p_{69}+p_{101}+p_{109}+p_{125}+1\Big)+\Big(k_{37}+k_{45}+k_{61}+k_{69}+k_{101}+k_{109}+k_{125}+p_{37}+p_{45}+p_{61}+p_{69}+p_{101}+p_{109}+p_{125}\Big)\Big(k_{33}+p_1+p_9+p_{25}+p_{33}+p_{65}+p_{73}+p_{89}+\Big(k_{98}+p_2+p_{10}+p_{26}+p_{66}+p_{74}+p_{90}+p_{98}+1\Big)\Big(k_{37}+k_{45}+k_{61}+k_{69}+k_{101}+k_{109}+k_{125}+p_{37}+p_{45}+p_{61}+p_{69}+p_{101}+p_{109}+p_{125}+1\Big)\Big)\Big)\Big(k_{33}+p_1+p_9+p_{25}+p_{33}+p_{65}+p_{73}+p_{89}+\Big(k_{98}+p_2+p_{10}+p_{26}+p_{66}+p_{74}+p_{90}+p_{98}+1\Big)\Big(k_{37}+k_{45}+k_{61}+k_{69}+k_{101}+k_{109}+k_{125}+p_{37}+p_{45}+p_{61}+p_{69}+p_{101}+p_{109}+p_{125}+1\Big)+\Big(k_0+k_{32}+k_{40}+k_{56}+k_{96}+k_{104}+k_{120}+p_0+p_{32}+p_{40}+p_{56}+p_{96}+p_{104}+p_{120}+\Big(k_{37}+k_{45}+k_{61}+k_{69}+k_{101}+k_{109}+k_{125}+p_{37}+p_{45}+p_{61}+p_{69}+p_{101}+p_{109}+p_{125}\Big)\Big(k_{33}+p_1+p_9+p_{25}+p_{33}+p_{65}+p_{73}+p_{89}+\Big(k_{98}+p_2+p_{10}+p_{26}+p_{66}+p_{74}+p_{90}+p_{98}+1\Big)\Big(k_{37}+k_{45}+k_{61}+k_{69}+k_{101}+k_{109}+k_{125}+p_{37}+p_{45}+p_{61}+p_{69}+p_{101}+p_{109}+p_{125}+1\Big)\Big)+1\Big)\Big(k_0+k_{32}+k_{33}+k_{37}+k_{40}+k_{45}+k_{56}+k_{61}+k_{69}+k_{96}+k_{98}+k_{101}+k_{104}+k_{109}+k_{120}+k_{125}+p_0+p_1+p_2+p_9+p_{10}+p_{25}+p_{26}+p_{32}+p_{33}+p_{37}+p_{40}+p_{45}+p_{56}+p_{61}+p_{65}+p_{66}+p_{69}+p_{73}+p_{74}+p_{89}+p_{90}+p_{96}+p_{98}+p_{101}+p_{104}+p_{109}+p_{120}+p_{125}+\Big(k_{98}+p_2+p_{10}+p_{26}+p_{66}+p_{74}+p_{90}+p_{98}+1\Big)\Big(k_{37}+k_{45}+k_{61}+k_{69}+k_{101}+k_{109}+k_{125}+p_{37}+p_{45}+p_{61}+p_{69}+p_{101}+p_{109}+p_{125}+1\Big)+\Big(k_{37}+k_{45}+k_{61}+k_{69}+k_{101}+k_{109}+k_{125}+p_{37}+p_{45}+p_{61}+p_{69}+p_{101}+p_{109}+p_{125}\Big)\Big(k_{33}+p_1+p_9+p_{25}+p_{33}+p_{65}+p_{73}+p_{89}+\Big(k_{98}+p_2+p_{10}+p_{26}+p_{66}+p_{74}+p_{90}+p_{98}+1\Big)\Big(k_{37}+k_{45}+k_{61}+k_{69}+k_{101}+k_{109}+k_{125}+p_{37}+p_{45}+p_{61}+p_{69}+p_{101}+p_{109}+p_{125}+1\Big)\Big)+1\Big)\Big).$

$b_{63} = \Big(k_{96}+p_0+p_8+p_{24}+p_{64}+p_{72}+p_{88}+p_{96}\Big)\Big(k_{35}+k_{43}+k_{59}+k_{67}+k_{99}+k_{107}+k_{123}+p_{35}+p_{43}+p_{59}+p_{67}+p_{99}+p_{107}+p_{123}+1\Big)+\Big(\Big(k_{35}+k_{43}+k_{59}+k_{67}+k_{99}+k_{107}+k_{123}+p_{35}+p_{43}+p_{59}+p_{67}+p_{99}+p_{107}+p_{123}\Big)\Big(\Big(k_{96}+p_0+p_8+p_{24}+p_{64}+p_{72}+p_{88}+p_{96}\Big)\Big(k_{35}+k_{43}+k_{59}+k_{67}+k_{99}+k_{107}+k_{123}+p_{35}+p_{43}+p_{59}+p_{67}+p_{99}+p_{107}+p_{123}+1\Big)+k_{63}+p_7+p_{23}+p_{31}+p_{63}+p_{71}+p_{87}+p_{95}\Big)+k_{30}+k_{38}+k_{54}+k_{62}+k_{102}+k_{118}+k_{126}+p_{30}+p_{38}+p_{54}+p_{62}+p_{102}+p_{118}+p_{126}+1\Big)\Big(\Big(k_{96}+p_0+p_8+p_{24}+p_{64}+p_{72}+p_{88}+p_{96}\Big)\Big(k_{35}+k_{43}+k_{59}+k_{67}+k_{99}+k_{107}+k_{123}+p_{35}+p_{43}+p_{59}+p_{67}+p_{99}+p_{107}+p_{123}+1\Big)+\Big(k_{35}+k_{43}+k_{59}+k_{67}+k_{99}+k_{107}+k_{123}+p_{35}+p_{43}+p_{59}+p_{67}+p_{99}+p_{107}+p_{123}\Big)\Big(\Big(k_{96}+p_0+p_8+p_{24}+p_{64}+p_{72}+p_{88}+p_{96}\Big)\Big(k_{35}+k_{43}+k_{59}+k_{67}+k_{99}+k_{107}+k_{123}+p_{35}+p_{43}+p_{59}+p_{67}+p_{99}+p_{107}+p_{123}+1\Big)+k_{63}+p_7+p_{23}+p_{31}+p_{63}+p_{71}+p_{87}+p_{95}\Big)+k_{30}+k_{35}+k_{38}+k_{43}+k_{54}+k_{59}+k_{62}+k_{63}+k_{67}+k_{96}+k_{99}+k_{102}+k_{107}+k_{118}+k_{123}+k_{126}+p_0+p_7+p_8+p_{23}+p_{24}+p_{30}+p_{31}+p_{35}+p_{38}+p_{43}+p_{54}+p_{59}+p_{62}+p_{63}+p_{64}+p_{67}+p_{71}+p_{72}+p_{87}+p_{88}+p_{95}+p_{96}+p_{99}+p_{102}+p_{107}+p_{118}+p_{123}+p_{126}\Big)+k_{63}+p_7+p_{23}+p_{31}+p_{63}+p_{71}+p_{87}+p_{95}.$

$b_{127} = \Big(k_{34}+k_{42}+k_{58}+k_{66}+k_{98}+k_{106}+k_{122}+p_{34}+p_{42}+p_{58}+p_{66}+p_{98}+p_{106}+p_{122}\Big)\Big(\Big(k_{34}+k_{42}+k_{58}+k_{66}+k_{98}+k_{106}+k_{122}+p_{34}+p_{42}+p_{58}+p_{66}+p_{98}+p_{106}+p_{122}+1\Big)\Big(k_{127}+p_7+p_{23}+p_{31}+p_{71}+p_{87}+p_{95}+p_{127}+1\Big)+k_{62}+p_6+p_{22}+p_{30}+p_{62}+p_{70}+p_{86}+p_{94}\Big)+k_{29}+k_{37}+k_{53}+k_{61}+k_{101}+k_{117}+k_{125}+p_{29}+p_{37}+p_{53}+p_{61}+p_{101}+p_{117}+p_{125}.$

Note:The correctness of these expressions are guaranteed by the correctness of the symbolic encryption code on which we have made a correctness test using the test vector in [24] and the encryption code for digital encryption and symbolic encryption are the same.

3.3. Algebraic expressions of the Hamming weight mod2 of the lower half and higher half of the output of the 1st round of NOEKEON

Based on the algebraic expressions of the output $B = b_0b_1...b_{127}$ of the 1st round,we can obtain the algebraic expressions of Hamming weight mod2 of the lower half and higher half of $B = b_0b_1...b_{127}$.

Let $h_l = b_{64} \oplus b_{65} \oplus ... \oplus b_{127}$,which is the Hamming weight mod2 of the lower 64 bits of the output of round 1,then by summing the expressions of $b_i, 64 \leq i \leq 127$,we have

$h_l = h_l(k_{32}, k_{33}, ..., k_{126}, k_{127}, p_0, p_1, ..., p_{127}) =$
$k_{64}+k_{65}+k_{66}+k_{67}+k_{68}+k_{69}+k_{70}+k_{71}+k_{72}+k_{73}+k_{74}+k_{75}+k_{76}+k_{77}+k_{78}+k_{79}+k_{80}+k_{81}+k_{82}+k_{83}+k_{84}+k_{85}+k_{86}+k_{87}+k_{88}+k_{89}+k_{90}+k_{91}+k_{92}+k_{93}+k_{94}+k_{95}+p_{64}+p_{65}+p_{66}+p_{67}+p_{68}+p_{69}+p_{70}+p_{71}+p_{72}+p_{73}+p_{74}+p_{75}+p_{76}+p_{77}+p_{78}+p_{79}+p_{80}+p_{81}+p_{82}+p_{83}+p_{84}+p_{85}+p_{86}+p_{87}+p_{88}+p_{89}+p_{90}+p_{91}+p_{92}+p_{93}+p_{94}+p_{95}+\Big(k_{101}+p_5+p_{13}+p_{29}+p_{69}+p_{77}+p_{93}+p_{101}+1\Big)\Big(k_{32}+k_{40}+k_{48}+k_{72}+k_{96}+k_{104}+k_{112}+p_{32}+p_{40}+p_{48}+p_{72}+p_{96}+p_{104}+p_{112}+1\Big)+\Big(k_{102}+p_6+p_{14}+p_{30}+p_{70}+p_{78}+p_{94}+p_{102}+1\Big)\Big(k_{33}+k_{41}+k_{49}+k_{73}+k_{97}+k_{105}+k_{113}+p_{33}+p_{41}+p_{49}+p_{73}+p_{97}+p_{105}+p_{113}+1\Big)+\Big(k_{103}+p_7+p_{15}+p_{31}+p_{71}+p_{79}+p_{95}+p_{103}+1\Big)\Big(k_{34}+k_{42}+k_{50}+k_{74}+k_{98}+k_{106}+k_{114}+p_{34}+p_{42}+p_{50}+p_{74}+p_{98}+p_{106}+p_{114}+1\Big)+\Big(k_{104}+p_0+p_8+p_{16}+p_{64}+p_{72}+p_{80}+p_{104}+1\Big)\Big(k_{35}+k_{43}+k_{51}+k_{75}+k_{99}+k_{107}+k_{115}+p_{35}+p_{43}+p_{51}+p_{75}+p_{99}+p_{107}+p_{115}+1\Big)+\Big(k_{105}+p_1+p_9+p_{17}+p_{65}+p_{73}+p_{81}+p_{105}+1\Big)\Big(k_{36}+k_{44}+k_{52}+k_{76}+k_{100}+$

$k_{108}+k_{116}+p_{36}+p_{44}+p_{52}+p_{76}+p_{100}+p_{108}+p_{116}+1\big)+\big(k_{106}+p_2+p_{10}+p_{18}+p_{66}+p_{74}+p_{82}+p_{106}+1\big)\big(k_{37}+k_{45}+k_{53}+k_{77}+k_{101}+k_{109}+k_{117}+p_{37}+p_{45}+p_{53}+p_{77}+p_{101}+p_{109}+p_{117}+1\big)+\big(k_{107}+p_3+p_{11}+p_{19}+p_{67}+p_{75}+p_{83}+p_{107}+1\big)\big(k_{38}+k_{46}+k_{54}+k_{78}+k_{102}+k_{110}+k_{118}+p_{38}+p_{46}+p_{54}+p_{78}+p_{102}+p_{110}+p_{118}+1\big)+\big(k_{108}+p_4+p_{12}+p_{20}+p_{68}+p_{76}+p_{84}+p_{108}+1\big)\big(k_{39}+k_{47}+k_{55}+k_{79}+k_{103}+k_{111}+k_{119}+p_{39}+p_{47}+p_{55}+p_{79}+p_{103}+p_{111}+p_{119}+1\big)+\big(k_{117}+p_{13}+p_{21}+p_{29}+p_{77}+p_{85}+p_{93}+p_{117}+1\big)\big(k_{32}+k_{48}+k_{56}+k_{88}+k_{96}+k_{112}+k_{120}+p_{32}+p_{48}+p_{56}+p_{88}+p_{96}+p_{112}+p_{120}+1\big)+\big(k_{109}+p_5+p_{13}+p_{21}+p_{69}+p_{77}+p_{85}+p_{109}+1\big)\big(k_{40}+k_{48}+k_{56}+k_{80}+k_{104}+k_{112}+k_{120}+p_{40}+p_{48}+p_{56}+p_{80}+p_{104}+p_{112}+p_{120}+1\big)+\big(k_{118}+p_{14}+p_{22}+p_{30}+p_{78}+p_{86}+p_{94}+p_{118}+1\big)\big(k_{33}+k_{49}+k_{57}+k_{89}+k_{97}+k_{113}+k_{121}+p_{33}+p_{49}+p_{57}+p_{89}+p_{97}+p_{113}+p_{121}+1\big)+\big(k_{110}+p_6+p_{14}+p_{22}+p_{70}+p_{78}+p_{86}+p_{110}+1\big)\big(k_{41}+k_{49}+k_{57}+k_{81}+k_{105}+k_{113}+k_{121}+p_{41}+p_{49}+p_{57}+p_{81}+p_{105}+p_{113}+p_{121}+1\big)+\big(k_{119}+p_{15}+p_{23}+p_{31}+p_{79}+p_{87}+p_{95}+p_{119}+1\big)\big(k_{34}+k_{50}+k_{58}+k_{90}+k_{98}+k_{114}+k_{122}+p_{34}+p_{50}+p_{58}+p_{90}+p_{98}+p_{114}+p_{122}+1\big)+\big(k_{111}+p_7+p_{15}+p_{23}+p_{71}+p_{79}+p_{87}+p_{111}+1\big)\big(k_{42}+k_{50}+k_{58}+k_{82}+k_{106}+k_{114}+k_{122}+p_{42}+p_{50}+p_{58}+p_{82}+p_{106}+p_{114}+p_{122}+1\big)+\big(k_{96}+p_0+p_8+p_{24}+p_{64}+p_{72}+p_{88}+p_{96}\big)\big(k_{35}+k_{43}+k_{59}+k_{67}+k_{99}+k_{107}+k_{123}+p_{35}+p_{43}+p_{59}+p_{67}+p_{99}+p_{107}+p_{123}+1\big)+\big(k_{120}+p_0+p_{16}+p_{24}+p_{64}+p_{80}+p_{88}+p_{120}\big)\big(k_{35}+k_{51}+k_{59}+k_{91}+k_{99}+k_{115}+k_{123}+p_{35}+p_{51}+p_{59}+p_{91}+p_{99}+p_{115}+p_{123}+1\big)+\big(k_{112}+p_8+p_{16}+p_{24}+p_{72}+p_{80}+p_{88}+p_{112}\big)\big(k_{43}+k_{51}+k_{59}+k_{83}+k_{107}+k_{115}+k_{123}+p_{43}+p_{51}+p_{59}+p_{83}+p_{107}+p_{115}+p_{123}+1\big)+\big(k_{97}+p_1+p_9+p_{25}+p_{65}+p_{73}+p_{89}+p_{97}+1\big)\big(k_{36}+k_{44}+k_{60}+k_{68}+k_{100}+k_{108}+k_{124}+p_{36}+p_{44}+p_{60}+p_{68}+p_{100}+p_{108}+p_{124}+1\big)+\big(k_{121}+p_1+p_{17}+p_{25}+p_{65}+p_{81}+p_{89}+p_{121}+1\big)\big(k_{36}+k_{52}+k_{60}+k_{92}+k_{100}+k_{116}+k_{124}+p_{36}+p_{52}+p_{60}+p_{92}+p_{100}+p_{116}+p_{124}+1\big)+\big(k_{113}+p_9+p_{17}+p_{25}+p_{73}+p_{81}+p_{89}+p_{113}+1\big)\big(k_{44}+k_{52}+k_{60}+k_{84}+k_{108}+k_{116}+k_{124}+p_{44}+p_{52}+p_{60}+p_{84}+p_{108}+p_{116}+p_{124}+1\big)+\big(k_{32}+k_{40}+k_{56}+k_{64}+k_{96}+k_{104}+k_{120}+p_{32}+p_{40}+p_{56}+p_{64}+p_{96}+p_{104}+p_{120}+1\big)\big(k_{125}+p_5+p_{21}+p_{29}+p_{69}+p_{85}+p_{93}+p_{125}+1\big)+\big(k_{98}+p_2+p_{10}+p_{26}+p_{66}+p_{74}+p_{90}+p_{98}+1\big)\big(k_{37}+k_{45}+k_{61}+k_{69}+k_{101}+k_{109}+k_{125}+p_{37}+p_{45}+p_{61}+p_{69}+p_{101}+p_{109}+p_{125}+1\big)+\big(k_{122}+p_2+p_{18}+p_{26}+p_{66}+p_{82}+p_{90}+p_{122}+1\big)\big(k_{37}+k_{53}+k_{61}+k_{93}+k_{101}+k_{117}+k_{125}+p_{37}+p_{53}+p_{61}+p_{93}+p_{101}+p_{117}+p_{125}+1\big)+\big(k_{114}+p_{10}+p_{18}+p_{26}+p_{74}+p_{82}+p_{90}+p_{114}+1\big)\big(k_{45}+k_{53}+k_{61}+k_{85}+k_{109}+k_{117}+k_{125}+p_{45}+p_{53}+p_{61}+p_{85}+p_{109}+p_{117}+p_{125}+1\big)+\big(k_{33}+k_{41}+k_{57}+k_{65}+k_{97}+k_{105}+k_{121}+p_{33}+p_{41}+p_{57}+p_{65}+p_{97}+p_{105}+p_{121}+1\big)\big(k_{126}+p_6+p_{22}+p_{30}+p_{70}+p_{86}+p_{94}+p_{126}+1\big)+\big(k_{99}+p_3+p_{11}+p_{27}+p_{67}+p_{75}+p_{91}+p_{99}+1\big)\big(k_{38}+k_{46}+k_{62}+k_{70}+k_{102}+k_{110}+k_{126}+p_{38}+p_{46}+p_{62}+p_{70}+p_{102}+p_{110}+p_{126}+1\big)+\big(k_{123}+p_3+p_{19}+p_{27}+p_{67}+p_{83}+p_{91}+p_{123}+1\big)\big(k_{38}+k_{54}+k_{62}+k_{94}+k_{102}+k_{118}+k_{126}+p_{38}+p_{54}+p_{62}+p_{94}+p_{102}+p_{118}+p_{126}+1\big)+\big(k_{115}+p_{11}+p_{19}+p_{27}+p_{75}+p_{83}+p_{91}+p_{115}+1\big)\big(k_{46}+k_{54}+k_{62}+k_{86}+k_{110}+k_{118}+k_{126}+p_{46}+p_{54}+p_{62}+p_{86}+p_{110}+p_{118}+p_{126}+1\big)+\big(k_{34}+k_{42}+k_{58}+k_{66}+k_{98}+k_{106}+k_{122}+p_{34}+p_{42}+p_{58}+p_{66}+p_{98}+p_{106}+p_{122}+1\big)\big(k_{127}+p_7+p_{23}+p_{31}+p_{71}+p_{87}+p_{95}+p_{127}+1\big)+\big(k_{100}+p_4+p_{12}+p_{28}+p_{68}+p_{76}+p_{92}+p_{100}+1\big)\big(k_{39}+k_{47}+k_{63}+k_{71}+k_{103}+k_{111}+k_{127}+p_{39}+p_{47}+p_{63}+p_{71}+p_{103}+p_{111}+p_{127}+1\big)+\big(k_{124}+p_4+p_{20}+p_{28}+p_{68}+p_{84}+p_{92}+p_{124}+1\big)\big(k_{39}+k_{55}+k_{63}+k_{95}+k_{103}+k_{119}+k_{127}+p_{39}+p_{55}+p_{63}+p_{95}+p_{103}+p_{119}+p_{127}+1\big)+\big(k_{116}+p_{12}+p_{20}+p_{28}+p_{76}+p_{84}+p_{92}+p_{116}+1\big)\big(k_{47}+k_{55}+k_{63}+k_{87}+k_{111}+k_{119}+k_{127}+p_{47}+p_{55}+p_{63}+p_{87}+p_{111}+p_{119}+p_{127}+1\big).$

Let $h_h = b_0 \oplus b_1 \oplus ... \oplus b_{63}$,which is the Hamming weight mod2 of the higher 64 bits of the output of round 1.Similarly,we can also obtain its expression,which is too complex to be given here.Each plaintext variable and key variable appears in the expression.i.e. we have

$h_h = h_h(k_0, k_1, ..., k_{126}, k_{127}, p_0, p_1, ..., p_{127})$.

3.4. Improved side channel attack on NOEKEON by MFCSCA

In this section,we give our MFCSCA attack on NOEKEON,which improves the work of Shekh Faisal Abdul-Latip et al in [9]. We use the Hamming weight leakage model.We suppose that the Hamming weight of the lower 64 bits $b_{64}b_{65}...b_{127}$ and the higher 64 bits $b_0b_1...b_{63}$ of the output of the first round can be obtained without error.We give a simulation MFCSCA attack on NOEKEON,which shows that the 128-bit key of NOEKEON can be recovered in 10 seconds on a PC with 2.6GHZ cpu and 8G RAM,with 99 known plaintexts.

The simulation attack contains two parts.

1.Recovering $k_{96}, k_{97}, ..., k_{127}$ with $n \geq 64$ known plaintexs,using the expressions of $h_l = h_l(k_{32}, k_{33}, ..., k_{126}, k_{127}, p_0, p_1, ..., p_{127})$ in section 3.3. The simulation steps are as follows:

(1)Random choose a key $K = k_0k_1...k_{127}$.

(2)Random choose n known plaintexts.Encrypt each plaintext with the key in (1) and output the Hamming weight(mod 2) of the **lower** 64 bits of the output of the 1st round.

(3)Using the n Hamming weights(mod 2) and the expressions of $h_l = h_l(k_{32}, k_{33}, ..., k_{126}, k_{127}, p_0, p_1, ..., p_{127})$ in section 3.3,we can obtain n equations with $k_{32}, k_{33}, ..., k_{126}, k_{127}$ as variables.

(4)Output these equations to a text file mg.txt,of which Magma v2.12-16 can find the Gröbner basis.Here we use the lexicographical order for finding the Gröbner basis in Magma.

(5)Use the Magma command "load mg.txt" to find the Gröbner basis of the equation system.

The steps (1)-(4) are simulated in Mathematica and step (5) in Magma.

We have made 100 simulations of (1)-(5) with $n = 70$.Each time Magma can find 64 bits key,of which the exact value of $k_{96}, k_{97}, ..., k_{127}$ can be obtained.The average running time of step (1)-(4) in Mathematica is 3.5 seconds and the average running time of step (5) in Magma is less than 1 second on a PC with 2.6GHZ CPU and 8G RAM.Hence the average time of recovering $k_{96}, k_{97}, ..., k_{127}$ is 4.5 seconds.

2.Recovering $k_0, k_1, ..., k_{95}$ with another m(i.e. total $m + n$,denoted by u) known plaintexs,using the expressions of $h_h = h_h(k_0, k_1, ..., k_{126}, k_{127}, p_0, p_1, ..., p_{127})$ in section 3.3. Now the variables $k_{96}, k_{97}, ..., k_{127}$ are replaced by the value obtained in 1. The simulation steps are as follows:

(1)Random choose a key $K = k_0k_1...k_{127}$.

(2)Random choose u known plaintexts.Encrypt each plaintext with the key in (1) and output the Hamming weight(mod 2) of the **higher** 64 bits of the output of the 1st round.

(3)Using the u Hamming weights(mod 2) and the expressions of $h_h = h_h(k_0, k_1, ..., k_{126}, k_{127}, p_0, p_1, ..., p_{127})$ in section 3.3,we can obtain u equations with $k_0, k_1, ..., k_{94}, k_{95}$ as variables(The variables $k_{96}, k_{97}, ..., k_{127}$ are replaced by the actual digital value in (1)).

(4)Output these equations to a text file mg.txt,of which Magma v2.12-16 can find the Gröbner basis.Here we use the lexicographical order for finding the Gröbner basis in Magma.

(5)Use the Magma command "load mg.txt" to find the Gröbner basis of the equation system.

The steps (1)-(4) are simulated in Mathematica and step (5) in Magma.

We have made 100 simulations of (1)-(5) with $u = 99$.Each time Magma can find the Gröbner basis of the equation system in (3).One running instance of the obtained Gröbner basis by Magma is:

$$[k_0+1, k_1, k_2, k_3, k_4, k_5+1, ..., k_{92}, k_{93}+1, k_{94}+1, k_{95}+1],$$

which means that $k_0 = 1, k_1 = 0, k_2 = 0, k_3 = 0, k_4 = 0, k_5 = 1, ..., k_{92} = 0, k_{93} = 1, k_{94} = 1, k_{95} = 1$.The average running time of step (1)-(4) in Mathematica is 4.8 seconds and the average running time of step (5) in Magma is 10.8 seconds on a PC with 2.6GHZ CPU and 8G RAM.Hence the average time of recovering $k_0, k_1, ..., k_{95}$ is 15.6 seconds and the total time of recovering the 128-bit key is 15.6+4.5=20.1 seconds.

Table 3: **Summary of side channel attack on NOEKEON**

Source	Leakage model	Data Complexity	Recovered Key
[9]	A	2^{10} CP	60 bits
this paper	B	99 KP	128 bits

Note:CP:Chosen Plaintext;KP:Known Plaintext;
A:single bit leakage of the output of the second round;
B:Hamming weight leakage of the lower 64 bits and higher 64 bits of the output of the 1st round;

We summarize our work and the known side channel attack on NOEKEON in Table 3. Our side channel attack improves upon the previous work of Shekh Faisal Abdul-Latip et al. from two aspects. First, we use the Hamming weight leakage model(we suppose the Hamming weight of the lower 64 bits and the higher 64 bits of the output of the first round can be obtained without error) which is a more relaxed leakage assumption, supported by many previously known practical results on side channel attacks, compared to the more challenging leakage assumption that the adversary has access to the "exact" value of the internal state bits as used by Shekh Faisal Abdul-Latip et al. Second, our attack has also a reduced complexity compared to that of Shekh Faisal Abdul-Latip et al. Namely, our attack of recovering the 128-bit key of NOEKEON has a time complexity 20.1 seconds on a PC and data complexity of 99 known plaintexts; whereas, that of Shekh Faisal Abdul-Latip et al. has time complexity of $O(2^{68})$ and needs about 2^{10} chosen plaintexts.

4. Conclusion

This paper proposes a new method of cryptanalysis of block cipher-MFCSCA,by combining the method of formal coding and side channel attack.The experimental result shows that if the Hamming weights of the output of the earlier rounds of a block cipher are leaked,then we can set up explicit equation system about the master key bits.By Gröbner basis-based method,we can find the master key.Although the diffusion is not enough,the Hamming weight of the entire state of earlier rounds

may contain all the information concerning the master key.If enough Hamming weights are leaked,than the master key can easily be recovered by equation solving technique.The following work is to use MFCSCA to attack other block ciphers.

Appendix A. Symbolic encryption code of NOEKEON in Mathematica platform

[1] Bogdanov, A., Kizhvatov, I., Pyshkin, A.: Algebraic Methods in Side-Channel Collision Attcks and Practical Collision Detection. In: Chowdhury, D.R., Rijmen, V., Das, A. (Eds.) INDOCRYPT 2008. LNCS, vol. 5365, pp. 251-265. Springer, Heidelberg (2008)

[2] Dinur, I., Shamir, A.: Side Channel Cube Attacks on Block Ciphers. Cryptology ePrint Archive, Report 2009/127 (2009), http://eprint.iacr.org/2009/127

[3] Mathieu Renauld and Franọis-Xavier Standaert. Algebraic Side-Channel Attacks. In Feng Bao, Moti Yung, Dongdai Lin, and Jiwu Jing, editors, Inscrypt, volume 6151 of Lecture Notes in Computer Science, pages 393-410. Springer, 2009.

[4] Mathieu Renauld, Franọis-Xavier Standaert, and Nicolas Veyrat-Charvillon. Algebraic Side-Channel Attacks on the AES: Why Time also Matters in DPA. In Christophe Clavier and Kris Gaj, editors, CHES, volume 5747 of Lecture Notes in Computer Science, pages 97-111. Springer, 2009.

[5] Yang, L., Wang, M., Qiao, S.: Side Channel Cube Attack on PRESENT. In: Garay, J.A., Miyaji, A., Otsuka, A. (Eds.) CANS 2009. LNCS, vol. 5888, pp. 379-391. Springer, Heidelberg (2009)

[6] Shekh Faisal Abdul-Latip, Mohammad Reza Reyhanitabar, Willy Susilo and Jennifer Seberry.Extended Cubes: Enhancing the cube attack by Extracting Low-Degree Non-linear Equations. In Bruce Cheung, Lucas Chi Kwong Hui, Ravi Sandhu, Duncan S.Wong (Eds.): ASIACCS 2011, pp.296-305,2011.

[7] XinJie Zhao, Tao Wang and ShiZe Guo: Improved Side Channel Cube Attacks on PRESENT. Cryptology ePrint Archive, Report 2011/165 (2011), http://eprint.iacr.org/2011/165

[8] Daemen, J., Peeters, M., Van Assche, G.,Rijmen, V.: Nessie Proposal: NOEKEON. First Open NESSIE Workshop (2000), http://gro.noekeon.org

[9] Shekh Faisal Abdul-Latip, Mohammad Reza Reyhanitabar, Willy Susilo, and Jennifer Seberry. On the Security of NOEKEON against Side Channel Cube Attacks. ISPEC 2010, LNCS 6047, 2010, pp. 45-55,2010.

[10] M. E. Hellman, R. Merkle, R. Schroppel, L. Washington, W. Diffe, S. Pohlig and P. Schweitzer, Results of an Initial Attempt to Cryptanalyze the NBS Data Encryption Standard, Technical Report, SEL 76-042, Stanford university, September 1976.

[11] Ingrid Schaumuller-Bichl, Zur Analyse des Data Encryption Standard und Synthese Verwandter Chiffriersysteme, Ph.D. Thesis, Linz university, May 1981.

[12] Ingrid Schaumuller-Bichl, Cryptanalysis of the Data Encryption Standard by the Method of Formal Coding, Lecture Notes in Computer Science, Cryptography, proceedings of the Workshop on Cryptography, Burg Feuerstein, Germany, March 29,April 2, pp. 235-255, 1982.

[13] Nicolas Courtois and Josef Pieprzyk: Cryptanalysis of Block Ciphers with Overdefined Systems of Equations, Asiacrypt 2002, LNCS 2501, pp.267-287, Springer.

[14] G. Bard, N. Courtois, C. Jefferson, Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers, Cryptology ePrint Archive, Report 2007/024.

[15] G. Ars, J.-C. Faugre, H. Imai, M. Kawazoe, M. Sugita, Comparison Between XL and Gröbner Basis Algorithms, in the proceedings of ASIACRYPT 2004, LNCS, vol 3329, pp 338-353, Jeju Island, Korea, December 2004.

[16] http://www.wolfram.com/mathematica/

[17] http://magma.maths.usyd.edu.au/magma/

[18] Bevan, R., Knudsen, R.: Ways to Enhance Differential Power Analysis. In: Lee, P.J, Lim, C.H. (Eds.) ICISC 2002. LNCS, vol. 2587, pp. 327-342. Springer, Heidleberg (2003)

[19] Clavier, C., Coron, J.S., Dabbous, N.: Differential Power Analysis in the Presence of Hardware Countermeasures. In: Koc, C.K., Paar, C. (Eds.) CHES 2000. LNCS, vol. 1965, pp. 252-263. Springer, Heidelberg (2000)

[20] Coron, J.S., Kocher, P., Naccache, D.: Statistics and Secret Leakage. In: Frankel, Y. (Ed.) FC 2000. LNCS, vol. 1962, pp. 157-173. Springer, Heidelberg (2001)

[21] Akkar, M.L., Bevan, R., Dischamp, P., Moyart, D.: Power analysis, what is now possible... In: Okamoto, T. (Ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 489-502. Springer, Heidelberg (2000)

[22] Mayer-Sommer, R.: Smartly Analysis the Simplicity and the Power of Simple Power Analysis on Smartcards. In: Koc, C.K., Paar, C. (Eds.) CHES 2000. LNCS, vol. 1965, pp. 78-92. Springer, Heidelberg (2000)

[23] Biham, E.: New Types of Cryptanalytic Attacks Using Related Keys. In: Tor Helleseth (Ed.) EUROCRYPT 1993. LNCS, vol. 7, pp. 229-246. Springer, Heidel- berg (1994)

[24] http://gro.noekeon.org/Noekeon_ref.zip

```
theta[key_, a_] := Module[{K0, K1, K2, K3, a0, a1, a2, a3, temp},
   a0 = a[[1 ;; 32]]; a1 = a[[33 ;; 64]]; a2 = a[[65 ;; 96]]; a3 = a[[97 ;; 128]];
   K0 = key[[1 ;; 32]]; K1 = key[[33 ;; 64]]; K2 = key[[65 ;; 96]]; K3 = key[[97 ;; 128]];
   temp = a0 + a2; temp = temp + RotateLeft[temp, 8] + RotateRight[temp, 8];
   a1 = a1 + temp; a3 = a3 + temp; a0 = a0 + K0; a1 = a1 + K1; a2 = a2 + K2;
   a3 = a3 + K3; temp = a1 + a3; temp = temp + RotateLeft[temp, 8] + RotateRight[temp, 8];
   a0 = a0 + temp; a2 = a2 + temp; Return[{a0, a1, a2, a3} // Flatten]];
pai1[x_] := {x[[1 ;; 32]], RotateLeft[x[[33 ;; 64]], 1],
    RotateLeft[x[[65 ;; 96]], 5], RotateLeft[x[[97 ;; 128]], 2]} // Flatten;
gamma[a_] := Module[{A0, A1, A2, A3, tmp}, A0 = a[[1 ;; 32]];
   A1 = a[[33 ;; 64]]; A2 = a[[65 ;; 96]]; A3 = a[[97 ;; 128]];
   A1 = A1 + (A2 + 1) * (A3 + 1); A0 = A0 + A1 * A2; tmp = A3; A3 = A0; A0 = tmp;
   A2 = A2 + A0 + A1 + A3; A1 = A1 + (A2 + 1) * (A3 + 1);
   A0 = A0 + A2 * A1; Return[{A0, A1, A2, A3} // Flatten]];
pai2[x_] := {x[[1 ;; 32]], RotateRight[x[[33 ;; 64]], 1],
    RotateRight[x[[65 ;; 96]], 5], RotateRight[x[[97 ;; 128]], 2]} // Flatten;
(*round constant of NOEKEON*)
rc1 = {128, 27, 54, 108, 216, 171, 77, 154, 47, 94, 188, 99, 198, 151, 53, 106, 212};
rc = Table[Table[0, {32}], {17}];
For[r = 1, r ≤ 17, r++, rc[[r]][[25 ;; 32]] = IntegerDigits[rc1[[r]], 2, 8]];
encryption[m_, key_, s_: 0] := Module[{c, r},
   c = m;
   For[r = 1, r ≤ 16, r++,
    c[[1 ;; 32]] = c[[1 ;; 32]] + rc[[r]];
    c = theta[key, c]; c = pai1[c]; c = gamma[c];
    c = (c /. Power[aaa_, cc_Integer] → aaa) /. aa_Integer → Mod[aa, 2];
    c = pai2[c];
    c = (c /. Power[aaa_, cc_Integer] → aaa) /. aa_Integer → Mod[aa, 2];
    If[s == r, Return[c]]];
   c[[1 ;; 32]] = c[[1 ;; 32]] + rc[[17]]; c = theta[key, c];
   c = (c /. Power[aaa_, cc_Integer] → aaa) /. aa_Integer → Mod[aa, 2];
   Return[c]];
(*correcness test of noekeonEncryption,the test vectors are taken from [24]*)
m = Table[0, {128}]; k = Table[0, {128}];
c = encryption[m, k]; BaseForm[FromDigits[c, 2], 16]
m = Table[1, {128}]; k = Table[1, {128}];
c = encryption[m, k]; BaseForm[FromDigits[c, 2], 16]
k = IntegerDigits[16^^b1656851699e29fa24b70148503d2dfc , 2, 128];
m = IntegerDigits[16^^2a78421b87c7d0924f26113f1d1349b2 , 2, 128];
c = encryption[m, k]; BaseForm[FromDigits[c, 2], 16]
(*obtain the algebraic expressions of round 1*)
m = Table[ToExpression["p" <> ToString[i]], {i, 0, 127}];
k = Table[ToExpression["k" <> ToString[i]], {i, 0, 127}];
c = encryption[m, k, 1];
Print[c[[1]], ",", c[[64]], ",", c[[128]]]
```

Figure A.3: Mathematica symbolic encryption code of NOEKEON.