SIDE CHANNEL ATTACK BY USING HIDDEN MARKOV MODEL

Hoang Long Nguyen

01 Septempber 2004

Dept. of Computer Science,
University of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB,
United Kingdom.
hn2503@bristol.ac.uk

Contents

1 Introduction

Side Channel Attack is an attack that enables extraction of a secret key stored in a cryptographic device, such as Smart Card. In this attack, an attacker monitors the power consumption of the cryptographic device and then sketches a graph of power or energy emitted against time. From this graph, the attacker can reconstruct the sequence of operations executed in the device because we can easily distinguish between Addition and Doubling as executing Doubling always emits less energy (heat) than Addition. As a consequence the attacker can recover all the bits of the binary representation of the secret key.

For example: let's have a look at Binary Multiplication Method and I shall assume that the attacker can distinguish between Addition and Doubling.

```
Q = M
P = 0
For i = 1 to N
    if($k_i$ == 1) then P = P + Q
    Q = 2Q
return P
```

As the attacker can derive the output trace from the graph of power consumption against time, for instance: the output trace is ADDADD. Then he or she also can recover the key which is 10100 in this case.

In order to prevent this kind of attack, various randomised exponentiation algorithms have been proposed in [1],[4],[5],[6],[8],[9]. The purpose of this project is to look at these algorithms and then to analyse whether or not they are vulnerable to side channel attacks via input-driven Hidden Markov Model (IDHMM) proposed in [2] and [3].

The report is organised as follows. I describe and draw IDHMM for three algorithms that are not secure against IDHMM in part I. In part II, I discuss the differences as well as advantages and disadvantages of two IDHMMs. Finally, I explain why IDHMM can not be used to attack a number of randomised algorithms in part III.

Part I

Algorithms that are vulnerable to Side Channel Attack by using Hidden Markov Model

2 NAF recoding algorithm

2.1 Non Adjacent Form (NAF) recoding algorithm

The reason why we want to compute the NAF of a non-negative integer given in binary representation is that point multiplication algorithm can be speed ed up because the NAF representation has no adjacent non-zero digits or in another word the binary representation is sparse.

```
ALGORITHM I.1: Conversion to NAF
INPUT:   An integer k = $\sum_{j=0}^{l-1} k_j 2^j$, $k_j \in \{0,1\}$.
OUTPUT: NAF k = $\sum_{j=0}^{l} s_j 2^j$, $s_j \in \{-1,0,1\}$.
1. $c_0 = 0$.
2. For j = 0 to l do:
3.     $c_{j+1} = \lfloor (k_j + k_{j+1} + c_j)/2 \rfloor$   (assume $k_i = 0$ for i $\geq$ l),
4.     $s_j = k_j + c_j - 2c_{j+1}$.
5. Return $(s_l s_{l-1} ... s_0)$.
```

Proof
• $1 \geq s_i \geq -1$.
From Steps 3 and 4:

$$\begin{array}{llccc} s_j & = & k_j + c_j - 2\lfloor (k_j + c_j + k_{j+1})/2 \rfloor & & \\ and & & & & \\ (k_j + c_j + k_{j+1})/2 & \geq & \lfloor (k_j + c_j + k_{j+1})/2 \rfloor & \geq & \lfloor (k_j + c_j)/2 \rfloor \\ & & & & \\ therefore & & & & \\ s_j & \geq & -k_{j+1} & \geq & -1 \\ s_j & \leq & k_j + c_j - 2\lfloor (k_j + c_j)/2 \rfloor & \leq & 1 \end{array}$$

• If $s_i = -1$ then $s_{i+1} = 0$
From Step 4: $s_i = k_j + c_j - 2c_{j+1}$
Therefore both c_{j+1} and $(k_j + c_j)$ are equal to 1.

$$\begin{array}{lcc}
c_{j+1} & = & \lfloor (k_j + k_{j+1} + c_j)/2 \rfloor \\
 & = & \lfloor (1 + k_{j+1})/2 \rfloor \\
 & = & 1 \\
Therefore & & \\
k_{j+1} & = & 1.
\end{array}$$

$$\begin{array}{lcc}
c_{j+2} & = & \lfloor (k_{j+1} + k_{j+2} + c_{j+1})/2 \rfloor \\
 & = & \lfloor (1 + k_{j+2} + 1)/2 \rfloor \\
 & = & 1 \\
Therefore & & \\
s_{i+1} & = & k_j + c_j - 2c_{j+2} \\
s_{i+1} & = & 0
\end{array}$$

• If $s_i = 1$ then $s_{i+1} = 0$
From Step 4: $s_i = k_j + c_j - 2c_{j+1}$
Therefore $c_{j+1} = 0$ and $(k_j + c_j) = 1$.

$$\begin{array}{lcc}
c_{j+1} & = & \lfloor (k_j + k_{j+1} + c_j)/2 \rfloor \\
 & = & 0 \\
therefore & & \\
k_{i+1} & = & 0
\end{array}$$

$$\begin{array}{lcc}
c_{j+2} & = & \lfloor (k_{j+1} + k_{j+2} + c_{j+1})/2 \rfloor \\
 & = & \lfloor (k_{j+2})/2 \rfloor \\
 & = & 0
\end{array}$$

$$\begin{array}{lcc}
s_{i+1} & = & k_{j+1} + c_{j+1} - 2c_{j+2} \\
 & = & 0
\end{array}$$

• s = $\sum_{j=0}^{l} s_i 2^j$ = k = $\sum_{j=0}^{l-1} k_i 2^j$

$$\begin{array}{lcl}
s & = & \sum_{j=0}^{l} s_j 2^j \\
 & = & \sum_{j=0}^{l} (k_j + c_j - 2c_{j+1}) 2^j \\
 & = & \sum_{j=0}^{l} k_j 2^j + \sum_{j=0}^{l} c_j 2^j - 2\sum_{j=0}^{l} c_{j+1} 2^j \\
 & = & \sum_{j=0}^{l} k_j 2^j + \sum_{j=0}^{l} c_j 2^j - \sum_{j=0}^{l} c_{j+1} 2^{j+1} \\
 & = & \sum_{j=0}^{l} k_j 2^j + \sum_{j=0}^{l} c_j 2^j - \sum_{j=1}^{l+1} c_j 2^j \\
 & = & \sum_{j=0}^{l} k_j 2^j + c_0 - c_{l+1} 2^{l+1} \\
 & = & \sum_{j=0}^{l} k_j 2^j + 0 - \lfloor (k_l + k_{l+1} + c_l)/2 \rfloor \\
 & = & \sum_{j=0}^{l} k_j 2^j - \lfloor (0 + 0 + c_l)/2 \rfloor \\
 & = & \sum_{j=0}^{l} k_j 2^j \\
 & = & k
\end{array}$$

2.2 Description of the randomised algorithm

In order to prevent DPA, a randomised NAF recoding algorithm was first proposed in 2003 by Jae Cheol Ha and Sang Jae Moon [1]. The randomised algorithm is built based on the ideas of NAF algorithm but the binary representation

Input			**Output**	
k_{i+1}	k_i	c_i	c_{i+1}	d_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	0
1	0	1	1	-1
1	1	0	1	-1
1	1	1	1	0

Table 1: NAF recoding method

of the output is not always in the form of NAF. However the density of non zero bit is still low (sparse).

As you can notice that we use auxiliary carry c_{i+1} in NAF recoding algorithm and c_{i+1} is the $(i+1)_{th}$ carry with c_0 and s_i is the i^{th} NAF digit. So that the concatenation:

$$c_{i+1}s_i = c_{i+1}2^1 + s_i 2^0 \ .$$

And if $c_{i+1}s_i = 01$ then alternatively we can represent it as $1\bar{1}$ and it is where random factor comes in as follows.

If we have $k_{i+1}k_ic_i = 001$ then from steps 3 and 4, we have $c_{i+1}s_i = 01$. So if a random bit r_i is set to 0 then $c_{i+1}s_i = 01$. Otherwise if the random bit r_i is set to 1 then $c_{i+1}s_i = 1\bar{1}$.

Apart from the cases when $k_{i+1}k_ic_i = 001, k_{i+1}k_ic_i = 010, k_{i+1}k_ic_i = 101, k_{i+1}k_ic_i = 110$ the NAF digit is independent of random bit.

You can see more details about each case in the tables 1 and 2.

2.3 Explanation of the State Diagram of the Randomised Algorithm

The algorithm is implemented from Right to Left or from Least Significant Bit (LSB) to Most Significant Bit (MSB).

The state diagram is in the form of Hidden Markov Model (HDD) described in [2] and [3]. However, there is one difference between this diagram and normal diagram. The difference is that at the very beginning, this model always guesses the first two least significant bits but after that it only guesses one bit at a time. The reason is explained as follows.

Input				**Output**		
k_{i+1}	k_i	c_i	r_i	c_{i+1}	d_i	Remarks
0	0	0	0	0	0	NAF
0	0	0	1	0	0	NAF
0	0	1	0	0	1	NAF
0	0	1	1	1	-1	AF
0	1	0	0	0	1	NAF
0	1	0	1	1	-1	AF
0	1	1	0	1	0	NAF
0	1	1	1	1	0	NAF
1	0	0	0	0	0	NAF
1	0	0	1	0	0	NAF
1	0	1	0	1	-1	NAF
1	0	1	1	0	1	AF
1	1	0	0	1	-1	NAF
1	1	1	1	0	1	AF
1	1	1	0	1	0	NAF
1	1	1	1	1	0	NAF

Table 2: Random NAF recoding method

As you can see from the original NAF recoding algorithm. In order to find the i^{th} bit, we need to know the i^{th} and $(i+1)^{th}$ bits of the binary representation of the input number. As a result, at the very beginning, to be able to match the first operation (either Doubling or Addition) we have to guess the first two least significant bits.

However, after the first stage, we continue to match the second NAF digit with the 2^{nd} and 3^{rd} input digit. But we have 2^{nd} bit in our pocket from the previous stage already that is why we only have to guess one extra bit from now on.

To summarise what I have said so far: In order to match s_i (for i $\geq$ 1), we need to guess k_i and k_{i+1} but we have k_i from the previous iteration (matching s_{i-1}) therefore we only need to guess k_{i+1}.

Transition	Key Bits	Probability
$q_1^{(n)} \to q_{10}^{(n+2)}$	00	1
$q_1^{(n)} \to q_{11}^{(n+2)}$	10	1
$q_1^{(n)} \to q_2^{(n+2)}$	01	1
$q_1^{(n)} \to q_3^{(n+2)}$	11	1
$q_2^{(n)} \to q_{13}^{(n)}$	n/a	1
$q_3^{(n)} \to q_{12}^{(n)}$	n/a	1
$q_4^{(n)} \to q_{10}^{(n)}$	n/a	1
$q_5^{(n)} \to q_{11}^{(n)}$	n/a	1
$q_6^{(n)} \to q_{14}^{(n)}$	n/a	1
$q_7^{(n)} \to q_{15}^{(n)}$	n/a	1
$q_8^{(n)} \to q_7^{(n+1)}$	1	1/2
$q_8^{(n)} \to q_5^{(n+1)}$	1	1/2
$q_8^{(n)} \to q_4^{(n+1)}$	0	1/2
$q_8^{(n)} \to q_6^{(n+1)}$	0	1/2
$q_9^{(n)} \to q_9^{(n+1)}$	1	1
$q_9^{(n)} \to q_8^{(n+1)}$	0	1
$q_{10}^{(n)} \to q_{10}^{(n+1)}$	0	1
$q_{10}^{(n)} \to q_{11}^{(n+1)}$	1	1
$q_{11}^{(n)} \to q_2^{(n+1)}$	0	1
$q_{11}^{(n)} \to q_3^{(n+1)}$	1	1
$q_{12}^{(n)} \to q_2^{(n+1)}$	0	1/2
$q_{12}^{(n)} \to q_8^{(n+1)}$	0	1/2
$q_{12}^{(n)} \to q_3^{(n+1)}$	1	1/2
$q_{12}^{(n)} \to q_9^{(n+1)}$	1	1/2
$q_{13}^{(n)} \to q_4^{(n+1)}$	0	1/4
$q_{13}^{(n)} \to q_6^{(n+1)}$	0	1/4
$q_{13}^{(n)} \to q_{10}^{(n+1)}$	0	1/2
$q_{13}^{(n)} \to q_5^{(n+1)}$	1	1/4
$q_{13}^{(n)} \to q_7^{(n+1)}$	1	1/4
$q_{13}^{(n)} \to q_{11}^{(n+1)}$	1	1/2
$q_{14}^{(n)} \to q_4^{(n+1)}$	0	1/2
$q_{14}^{(n)} \to q_6^{(n+1)}$	0	1/2
$q_{14}^{(n)} \to q_5^{(n+1)}$	1	1/2
$q_{14}^{(n)} \to q_7^{(n+1)}$	1	1/2
$q_{15}^{(n)} \to q_0^{(n+1)}$	0	1
$q_{15}^{(n)} \to q_1^{(n+1)}$	1	1

Table 3: The Transition probability table for the NAF recoding algorithm using Smart HMM.

3 Liardet-Smart Algorithm

3.1 Signed m-ary Window Decomposition

```
ALGORITHM II.1:Signed m-array Window decomposition
INPUT:   An integer k = $\sum_{j=0}^{l} 2^j, k_j \in \{0,1\}, k_l = 0$
OUTPUT: A sequence of pairs $\{(b_i, e_i)\}_{i=0}^{d-1}$
1. $d = 0, j = 0$.
2. While $j \leq l$ do:
3.     if $k_j = 0$ then j = j + 1
4.     else do:
5.         t = $\min\{l, j + r - 1\}$, $h_d = (k_t k_{t-1} ... k_j)_2$
6.         if $h_d > 2^{r-1}$ then do:
7.             $b_d = h_d - 2^r$,
8.             increment the number $(k_t k_{t-1} ... k_j)_2$ by 1
9.         $b_d = h_d$
10.        $e_d = j, d = d + 1, j = t + 1$.
11. Return the sequence $(b_0, e_0), (b_1, e_1), ..., (b_{d-1}, e_{d-1})$.
```

In this algorithm [12], the size of the window is always fixed or constant. As you might notice the window width does not affect the final result so that it can be varied randomly as suggested in [1].

3.2 Randomised Signed m-ary Window Decomposition

```
ALGORITHM II.21: Randomised Signed m-array Window decomposition
INPUT:   An integer k = $\sum_{j=0}^{l} 2^j, k_j \in \{0,1\}, k_l = 0$
OUTPUT: A sequence of pairs $\{(b_i, e_i)\}_{i=0}^{d-1}$
1. $d = 0, j = 0$.
2. While $j \leq l$ do:
3.     if $k_j = 0$ then j = j + 1
4.     else do:
5.         r $=_{random} \{1, ..., R\}$
6.         t = $\min\{l, j + r - 1\}$, $h_d = (k_t k_{t-1} ... k_j)_2$
7.         if $h_d > 2^{r-1}$ then do:
8.             $b_d = h_d - 2^r$,
9.             increment the number $(k_t k_{t-1} ... k_j)_2$ by 1
10.        $b_d = h_d$
11.        $e_d = j, d = d + 1, j = t + 1$.
12. Return the sequence $(b_0, e_0), (b_1, e_1), ..., (b_{d-1}, e_{d-1})$.
```

The only difference between this algorithm and the above is that window width is chosen randomly or in another word, the base is picked randomly from a set of power of two. Later on in this report, I shall write about another algorithm (MIST) which is only slightly different from the Smart-Liardet algorithm in the sense that the base is not necessary a power of two but can be any number chosen randomly.

At the next stage, the output of the Random Signed m-ary Window Decomposition which is a sequence of pairs $(b_0, e_0), (b_1, e_1), ..., (b_{d-1}, e_{d-1})$ will be the input of the Point Multiplication Algorithm as follows.

```
ALGORITHM II.22:Point Multiplication: Signed m-array Window
INPUT:    A point, P, and $\{(b_i, e_i)\}_{i=0}^{d-1}$ such that k = $\sum_{j=0}^{d-1} b_i 2^{e_i}$
OUTPUT: Q = [k]P
Precomputation
1. $P_1 = P, P_2 = [2]P$
2. For i = 1 to $2^{r-2} - 1$ do $P_{2i+1} = P_{2i-1} + P_2$
3. Q = $P_{b_{d-1}}$
Main loop
4. For i = d-2 to 0 by -1 do:
5.         Q = $[2^{e_{i+1}-e_i}]$Q.
6.         If $b_i > 0$ then $Q = Q + P_{-b_i}$,
7.         Else Q = Q - $P_{-b_i}$.
8. Q = $[2^{e_0}]$Q.
9.Return Q.
```

One may ask why cannot we merge the two above algorithms together to make the process more compact and faster. Well, it seems to me that we cannot do it because of the two reasons:

• Firstly, Random Decomposition Algorithm only works from right to left (or from LSB to MSB) and we can not reverse the process because there will be a problem with Step 7, 8 and 9 as it does not affect the output sequence any more when 1 is added to $(k_t k_{t-1}...k_j)_2$ if we run the algorithm from Left to Right.

• Secondly, Point Multiplication Algorithm works from Right to Left (MSB to LSB) and it can not be reversed, either. Assume that we run the algorithm from Left to Right and at the end of iteration i, we have:

$$Q = \sum_{j=0}^{j=i}((-1)^{b_j} P_{|b_j|} 2^{e_j})$$

It may works if the window width is always 1 but unfortunately the window size is variable so that we shall need one more variable to hold the value of 2^{e_i} and one more Multiplication which is $P_{|b_i|}2^{e_i}$ at every iteration.

3.3 Explanation of State Diagram of Smart and Liardet Algorithm

In this case, we assume that the maximum window size is 5 (R = 5) and the probability of all window widths are equal to one another: Pr(r = i) = 1/R for $1 \leq i \leq R$. The Hidden Markov Model guesses the secret key bit by bit from LSB to MST.

At the beginning, Doubling (D) is implemented repeatedly until the 1st non-zero bit is encountered. This behaviour is represented by State 1 of the State

Transition	Key Bits	Probability
$q_1^{(n)} \rightarrow q_1^{(n+1)}$	0	1
$q_1^{(n)} \rightarrow q_2^{(n+1)}$	1	1
$q_2^{(n)} \rightarrow q_{r+2}^{(n)}$	n/a	1
$q_3^{(n)} \rightarrow q_2^{(n+1)}$	0	1
$q_3^{(n)} \rightarrow q_3^{(n+1)}$	1	1
$q_i^{(n)} \rightarrow q_{i+1}^{(n+1)}$ for i = 4,...,r	0 or 1	1
$q_{r+1}^{(n)} \rightarrow q_1^{(n+1)}$	0	1
$q_{r+1}^{(n)} \rightarrow q_3^{(n+1)}$	1	1
$q_{r+2}^{(n)} \rightarrow q_1^{(n+1)}$	0	2/r
$q_{r+2}^{(n)} \rightarrow q_2^{(n+1)}$	1	1/r
$q_{r+2}^{(n)} \rightarrow q_3^{(n+1)}$	0	1/r
$q_{r+2}^{(n)} \rightarrow q_i^{(n+1)} for i = 4, ..., r+1$	0 or 1	1/r

Table 4: The Transition probability table for Liardet-Smart algorithm using Smart HMM.

Diagram.

As soon as the 1st non-zero bit is encountered, an Addition (A) will be implemented and then follows it there will be one or more D depending on the width of the window. This part is represented on State 2, 7 and 4..6.

- If the window size is 5 (R, maximum) then it jumps from State 7 to 4.
- If the window size is 4 (R-1) then it jumps from State 7 to 5.
- If the window size is 3 (R-2) then it jumps from State 7 to 6.
- The interesting case is when the window size is either 1 or 2:

if the next bit is zero then it jumps to State 1 with probability 2/r in both cases when window size is 1 or 2.

However if the next bit is 1 then there are two possibilities. When window width is 1 the next state is 2 (Addition) with probability 1/R otherwise as the last bit of the window is 1 as with other cases (when window size is greater than 2) it jumps to State 3.

Finally, the order of operation of State 1 and 3 are opposite to each other. The difference comes from the Step 7 and 9 of the Random Decomposition Algorithm when the last bit of the window is 1 then the behaviour of the next window depends on not only the next bit but also the previous window.

4 Binary Multiplication

4.1 Binary Multiplication Algorithm

```
ALGORITHM III.1: The Binary Algorithm for ECC scalar multiplication
Assume that k = $k_N k_{N-1} ... k_2 k_1$
INPUT: k, M
OUTPUT: k * M
Q = M
P = 0
For i = 1 to N
    if($k_i$ == 1) then P = P + Q
    Q = 2Q
return P
```

4.2 Randomised Binary Algorithm

Oswald and Aigner have proposed two randomised exponential algorithms [13] for scalar multiplication in ECC implementation. The algorithms are based on the randomisation of ADD-SUBTRACT chain. For example:

15P = P + 2(P + 2(P + 2P))

Alternatively 15P = 16P - P = 2(2(2(2P))) - P

More generally, a series of more than two 1's in the binary representation of k can be replaced by a block of 0's and a $\bar{1}$ (where $\bar{1}$ = -1).

$$01^a \rightarrow 10^{a-1}\bar{1} \text{ where } \bar{1} \text{ represents -1}$$

A second transformation proposed in [13] transforms an isolated 0's inside a block of 1's.

$$01^a01^b \rightarrow 10^a10^{b-1}\bar{1}$$

Proof:

$$\begin{aligned} 01^a01^b &= (2^a - 1)2^{b+1} + 2^b - 1 \\ &= 2^{a+b+1} - 2^{b+1} + 2^b - 1 \\ &= 2^{a+b+1} - 2^b - 1 \\ &= 10^a10^{b-1}\bar{1} \end{aligned}$$

In the Binary Multiplication Algorithm, at each step, we flip a coin to decide whether or not to apply the transformation. Below is the codes of two randomised algorithms.

ALGORITHM III.21: Randomised Addition-Subtraction Chains OA1

$01^a \rightarrow 10^{a-1}\bar{1}$

INPUT: k,M
OUTPUT: kM

```
Q = M
P = 0
State = 0
For i = 1 to N
    switch(State)
    case 0: if($k_i$ == 1) then P = P + Q, State = 1
            Q = 2Q
    case 1: if($k_i$ == 1) then
                b = ran-bit()
                if(b == 0) then P = P + Q
                else P = P - Q, State = 2
              else State = 0
              Q = 2Q
    case 2: if($k_i$ == 1) then
                b = ran-bit()
                if(b == 0) then Q = 2Q
                else Q = 2Q, P = P+Q, State = 1
              else P = P+Q, Q = 2Q, State = 0
return P
```

ALGORITHM III.22: Randomised Addition-Subtraction Chains OA2

$01^a01^b \rightarrow 10^a10^{b-1}\bar{1}$

Assume that k = $00k_{N-1}...k_2k_1k_0$

INPUT: k,M
OUTPUT: kM

```
Q = M
P = 0
State = 0
For i = 0 to N - 1
    switch(State)
    case 0: if($k_i$ == 1) then P = P + Q, State = 1
              Q = 2Q
    case 1: if($k_i$ == 1) then
                b = ran-bit()
                if(b == 0) then P = P - Q, State = 2
                else P = P + Q
              else State = 0
              Q = 2Q
    case 2: if($k_i$ == 0) then P = P - Q, State = 3
              Q = 2Q
    case 3: if($k_i$ == 1) then
                Q = 2Q
                if(b == 0) then P = P + Q, State = 1
              else P = P + Q, Q = 2Q, State = 0
return P
```

Transition	Key Bits	Probability
$q_1^{(n)} \to q_1^{(n+1)}$	0	1
$q_1^{(n)} \to q_2^{(n+1)}$	1	1
$q_2^{(n)} \to q_3^{(n)}$	n/a	1
$q_3^{(n)} \to q_2^{(n+1)}$	1	1/2
$q_3^{(n)} \to q_4^{(n+1)}$	1	1/2
$q_3^{(n)} \to q_1^{(n+1)}$	0	1
$q_4^{(n)} \to q_5^{(n)}$	n/a	1
$q_5^{(n)} \to q_5^{(n+1)}$	1	1/2
$q_5^{(n)} \to q_6^{(n+1)}$	1	1/2
$q_5^{(n)} \to q_8^{(n+1)}$	0	1
$q_6^{(n)} \to q_7^{(n)}$	n/a	1
$q_7^{(n)} \to q_2^{(n+1)}$	1	1/2
$q_7^{(n)} \to q_4^{(n+1)}$	1	1/2
$q_7^{(n)} \to q_1^{(n+1)}$	0	1
$q_8^{(n)} \to q_1^{(n)}$	n/a	1

Table 5: The Transition probability table for the OA1 binary exponentiation algorithm using Smart HMM. We have $Q^* = \{p_2, p_4, p_6\}$ and $Q^\perp = \{p_1, p_3, p_5, p_7\}$

4.3 Explanation of the State Diagram of the Randomised Binary Algorithm

The State Diagrams can be derived easily from the code of two randomised algorithms described above.

Transition	Key Bits	Probability
$q_0^{(n)} \to q_1^{(n+1)}$	0	1
$q_0^{(n)} \to q_2^{(n+1)}$	1	1
$q_1^{(n)} \to q_1^{(n+1)}$	0	1
$q_1^{(n)} \to q_2^{(n+1)}$	1	1
$q_2^{(n)} \to q_3^{(n)}$	n/a	1
$q_3^{(n)} \to q_2^{(n+1)}$	1	1/2
$q_3^{(n)} \to q_4^{(n+1)}$	1	1/2
$q_3^{(n)} \to q_1^{(n+1)}$	0	1
$q_4^{(n)} \to q_5^{(n)}$	n/a	1
$q_5^{(n)} \to q_5^{(n+1)}$	1	1
$q_5^{(n)} \to q_6^{(n+1)}$	0	1
$q_6^{(n)} \to q_7^{(n)}$	n/a	1
$q_7^{(n)} \to q_8^{(n+1)}$	0	1
$q_7^{(n)} \to q_9^{(n+1)}$	1	1/2
$q_7^{(n)} \to q_7^{(n+1)}$	1	1/2
$q_8^{(n)} \to q_1^{(n)}$	n/a	1
$q_9^{(n)} \to q_{10}^{(n)}$	n/a	1
$q_{10}^{(n)} \to q_1^{(n+1)}$	0	1
$q_{10}^{(n)} \to q_2^{(n+1)}$	1	1/2
$q_{10}^{(n)} \to q_4^{(n+1)}$	1	1/2

Table 6: The Transition probability table for the OA2 binary exponentiation algorithm using Smart HMM. We have $Q^* = \{p_2, p_4, p_6, p_8, p_9\}$ and $Q^{\perp} = \{p_1, p_3, p_5, p_7, p_{10}\}$

Part II
Comparison between two types of Hidden Markov Models (HMM)

What I am going to write in this part is something which I have noticed why working and playing with both kinds of HMMs that were originally proposed by Chris Karlop and David Wagner (I refer to this type of HMM as Karlop-Wagner HMM) in [2] and then by Nigel P Smart (I refer to this type of HMM as Smart HMM) in [3]. The conclusion is that the Smart HMM might be much more compact and easier to draw as well as time consuming that the Karlop-Wagner HMM. And there are two reasons for that as follows.

5 State Duplication

First of all, if you have read the paper by Chris Karlop and David Wagner [2], you might have noticed that, on page 22 of the paper, the authors said that the Karlop-Wagner HMM has to be faithful. So what it means is that there is one-to-one correspondence between an execution q and the key k used in that execution. Therefore there is no ambiguity in what bit annotated the corresponding directed edge that was taken from Q_i to Q_j: (Where Q_i and Q_j are states in the HMM)

$$\forall Q_i, Q_j \in Q \text{ if } Pr(Q_i, Q_j, 0) > 0 \Rightarrow Pr(Q_i, Q_j, 1) = 0 \text{ and vice versa.}$$

On the other hand, we do not need this condition in HMM suggested by Nigel Smart [3]. As a result, the number of states in the Karlop-Wagner HMM is often more than twice as many as the number of states in the Smart HMM.

For example: if we look at Liardet-Smart Algorithm.

• Using Smart HMM (Fig.2): the states Q4, Q5 and Q6 can take both 0 and 1 as input.

e.g: $Pr(Q_7, Q_4, 0) = Pr(Q_7, Q_4, 1) > 0$

And when the maximum window size is 5 then the number of states is 7. More generally, if the maximum widow width is R then the number of states in the State Diagram is 4 + (R - 2) = R + 2.

• Using Karlop-Wagner HMM (Fig.5): as you can see the two states Q2 and Q3 have the same functionality as single state Q7 in Fig.2 and similarly we have

Q4 + Q5 (Fig.5) = Q4 (Fig.2)

Q6 + Q7 (Fig.5) = Q5 (Fig.2)

Q8 + Q9 (Fig.5) = Q6 (Fig.2)

Extending of the Karlop-Wagner HMM to different maximum window sizes is trivial by removing or inserting new states such as Q10 and Q11..

We notice that in the Karlop-Wagner HMM for Smart-Liardet Algorithm, observable operations are placed on the edges rather on the states. Fortunately, edge annotated state machine can easily be transformed into a semantically equivalent state annotated machine by treating each edge as a state of the new HMM. However I am not going to do it here as the resulting model will be huge.

If we work out then when the maximum window width is 5 then there are 30 states (number of edges) in total. More generally, if the maximum window size is R then the number of states in the State Annotated Machine is 6 + 2[2(R-2) + 2(R-2)] = 8R - 10.

So that when R goes to infinity, the number of states in the Karlop-Wagner HMM is eight times as many as number of states in the Smart HMM.

6 Decomposition of observable sequence of operation

• In Smart HMM, each state only can carry with it either Doubling (D) or Addition (A) operation. As a result, it is very straight forward to decompose a observable sequence operation into a sequence of symbols from O (set of observable symbols).

• However in the Karlop-Wagner HMM, each state can carry with it any sequence of As or Ds and this is where we have a problem with deterministic decomposition.

For example: we look at OA1 algorithm where at each state, we flip a coin to decide whether or not to apply this transformation : $01^a = 10^{a-1}\bar{1}$

Assume that it would be convenient if our observable alphabet O = $\{D, AD, AAD\}^*$. From Fig.6(a) , the transition from Q2 to Q1 executes a D first and then an A, resulting in a DA output symbol corresponding to that key bit. This is undesirable as traces fail to be uniquely decode-able. For instance: DADD can be interpreted as either (DA,D,D) or (D,AD,D). To be able to fix the problem we have to relabel the DA transition from Q2 to Q1 to simply D and add a new state Q3 as in Fig.6(b) As a consequence the number of states is more than usual.
Furthermore, as each state can have more than one operation so that the number of possible states with same number of As and Ds is big a because there are many combination of Ds and As compared to only two symbols which are A and D in Smart HMM.

Transition	Key Bits	Probability
$q_1^{(n)} \rightarrow q_1^{(n+1)}$	0	1
$q_1^{(n)} \rightarrow q_4^{(n+1)}$	1	1
$q_2^{(n)} \rightarrow q_1^{(n+1)}$	0	1
$q_2^{(n)} \rightarrow q_4^{(n+1)}$	1	1
$q_3^{(n)} \rightarrow q_2^{(n+1)}$	0	1
$q_3^{(n)} \rightarrow q_3^{(n+1)}$	1	1/2
$q_3^{(n)} \rightarrow q_{11}^{(n+1)}$	1	1/2
$q_4^{(n)} \rightarrow q_5^{(n+1)}$	0	1
$q_4^{(n)} \rightarrow q_6^{(n+1)}$	1	1/2
$q_4^{(n)} \rightarrow q_9^{(n+1)}$	1	1/2
$q_5^{(n)} \rightarrow q_1^{(n+1)}$	0	1
$q_5^{(n)} \rightarrow q_4^{(n+1)}$	1	1
$q_6^{(n)} \rightarrow q_2^{(n+1)}$	0	1
$q_6^{(n)} \rightarrow q_3^{(n+1)}$	1	1/2
$q_6^{(n)} \rightarrow q_{11}^{(n+1)}$	1	1/2
$q_7^{(n)} \rightarrow q_1^{(n+1)}$	0	1
$q_7^{(n)} \rightarrow q_4^{(n+1)}$	1	1
$q_8^{(n)} \rightarrow q_5^{(n+1)}$	0	1
$q_8^{(n)} \rightarrow q_6^{(n+1)}$	1	1/2
$q_8^{(n)} \rightarrow q_9^{(n+1)}$	1	1/2
$q_9^{(n)} \rightarrow q_5^{(n+1)}$	0	1
$q_9^{(n)} \rightarrow q_6^{(n+1)}$	1	1/2
$q_9^{(n)} \rightarrow q_9^{(n+1)}$	1	1/2
$q_{10}^{(n)} \rightarrow q_2^{(n+1)}$	0	1
$q_{10}^{(n)} \rightarrow q_3^{(n+1)}$	1	1/2
$q_{10}^{(n)} \rightarrow q_{11}^{(n+1)}$	1	1/2
$q_{11}^{(n)} \rightarrow q_7^{(n+1)}$	0	1
$q_{11}^{(n)} \rightarrow q_8^{(n+1)}$	1	1/2
$q_{11}^{(n)} \rightarrow q_{10}^{(n+1)}$	1	1/2

Table 7: The Transition probability table for the OA1 binary exponentiation algorithm using Karlop-Wagner HMM.

Transition	Key Bits	Probability
$q_1^{(n)} \rightarrow q_1^{(n+1)}$	0	1
$q_1^{(n)} \rightarrow q_3^{(n+1)}$	1	1
$q_2^{(n)} \rightarrow q_{11}^{(n+1)}$	0	1
$q_2^{(n)} \rightarrow q_2^{(n+1)}$	1	1
$q_3^{(n)} \rightarrow q_4^{(n+1)}$	0	1
$q_3^{(n)} \rightarrow q_9^{(n+1)}$	1	1/2
$q_3^{(n)} \rightarrow q_5^{(n+1)}$	1	1/2
$q_4^{(n)} \rightarrow q_1^{(n+1)}$	0	1
$q_4^{(n)} \rightarrow q_3^{(n+1)}$	1	1
$q_5^{(n)} \rightarrow q_{11}^{(n+1)}$	0	1
$q_5^{(n)} \rightarrow q_2^{(n+1)}$	1	1
$q_6^{(n)} \rightarrow q_1^{(n+1)}$	0	1
$q_6^{(n)} \rightarrow q_3^{(n+1)}$	1	1
$q_7^{(n)} \rightarrow q_1^{(n+1)}$	0	1
$q_7^{(n)} \rightarrow q_3^{(n+1)}$	1	1
$q_8^{(n)} \rightarrow q_4^{(n+1)}$	0	1
$q_8^{(n)} \rightarrow q_9^{(n+1)}$	1	1/2
$q_8^{(n)} \rightarrow q_5^{(n+1)}$	1	1/2
$q_9^{(n)} \rightarrow q_4^{(n+1)}$	0	1
$q_9^{(n)} \rightarrow q_9^{(n+1)}$	1	1/2
$q_9^{(n)} \rightarrow q_5^{(n+1)}$	1	1/2
$q_{10}^{(n)} \rightarrow q_{11}^{(n+1)}$	0	1
$q_{10}^{(n)} \rightarrow q_2^{(n+1)}$	1	1
$q_{11}^{(n)} \rightarrow q_6^{(n+1)}$	0	1
$q_{11}^{(n)} \rightarrow q_{12}^{(n+1)}$	1	1/2
$q_{11}^{(n)} \rightarrow q_{13}^{(n+1)}$	1	1/2
$q_{12}^{(n)} \rightarrow q_7^{(n+1)}$	0	1
$q_{12}^{(n)} \rightarrow q_8^{(n+1)}$	1	1/2
$q_{12}^{(n)} \rightarrow q_{10}^{(n+1)}$	1	1/2
$q_{13}^{(n)} \rightarrow q_6^{(n+1)}$	0	1
$q_{13}^{(n)} \rightarrow q_{12}^{(n+1)}$	1	1/2
$q_{13}^{(n)} \rightarrow q_{13}^{(n+1)}$	1	1/2

Table 8: The Transition probability table for the OA2 binary exponentiation algorithm using Karlop-Wagner HMM.

Part III

Randomised Algorithms that are secure against HMM

In this part , I shall discuss a number of randomised algorithms that can not be attacked by Input Hidden Markov Model with explanation why not. In some cases we still can draw HMM for them but it does not mean they are vulnerable to side channel attack.

All algorithms discussed in this part are different ways of randomised decomposition.

7 Overlapping Window method and Simplifying MIST-Liardet-Smart algorithms

7.1 Simplifying MIST-Liardet-Smart algorithm

The first algorithm I am going to introduce is something that is similar to both MIST [5] and Liardet-Smart algorithms [1]. Below is the code for this algorithm:

```
ALGORITHM VI.1: Simple randomised Signed m-array Window decomposition
INPUT:  An integer k = $\sum_{j=0}^{l} 2^j, k_j \in \{0,1\}, k_l = 0$
OUTPUT: A sequence of pairs $\{(b_i, e_i)\}_{i=0}^{d-1}$
1. $d = 0, j = 0$.
2. While $j \le l$ do:
3.         r $=_{random} \{1, ..., R\}$
4.         t = $\min\{l, j + r - 1\}$, $b_d = (k_t k_{t-1} ... k_j)_2$
5.       $e_d = j, d = d + 1, j = t + 1$.
6. Return the sequence $(b_0, e_0), (b_1, e_1), ..., (b_{d-1}, e_{d-1})$.
```

This algorithm is similar to Liardet-Smart algorithm as the window size is chosen randomly therefore the base is always a power of two. However, in MIST, the base is not necessary a power of two as it can be any integer number.

Secondly, it is different from Liardet-Smart as it does not skip zero bit. Therefore the number of window will increase in the case when the binary representation of the input number is very sparse. It also does not add 1 to $(k_t k_{t-1} ... k_j)_2$ when the last bit of the previous window is non-zero so that b_i is always zero or positive number.

In general, the algorithm is very simple and it can be regarded as a small case of MIST algorithm.

7.2 Overlapping Window Method

This algorithm was proposed in [6].

In Overlapping window method, windows w_i and w_{i+1} are allowed to overlap.

$$\begin{aligned} k &= (k_{l-1}k_{l-2}...k_0)_2 \\ &= (...(w_0 * 2^{k-h_0} + w_1)...)2^{k-h_{s-1}} + w_s \end{aligned}$$

where:

k is the window size (constant throughout the algorithm).
s is the number of windows.
h_i is the overlapping bit length between w_i and w_{i+1}.
h_i for i $\in (0..s-1)$ is chosen randomly.

Therefore: sk - $\sum_{i=0}^{s-1} h_i = l$ (bit length of k)

For example:

$$\begin{aligned} d &= d_{l-1}...d_{l-k}d_{l-k-1}....d_{l-k-(k-h_0)}...d_0 \\ dt_0 &= (d_{l-1}...d_{l-k})_2 \end{aligned}$$

w_0 is picked randomly satisfying that $dt_0 - (2^{h_0} - 1) \leq w_0 \leq dt_0$

$$dt_1 = (dt_0 - w_0)2^{k-h_0} + (d_{l-k-1}....d_{l-k-(k-h_0)})_2$$

w_1 is picked randomly satisfying that $dt_1 - (2^{h_1} - 1) \leq w_1 \leq dt_1$

Now, assume that $d_{l-k-(k-h_0)}$ is the Least Significant Bit (LSB) of d then we terminate the algorithm here and $w_1 = dt_1$. Otherwise we reiterate the process until we get to the LSB of d.

Proof of the correctness of the algorithm:
Assume that $d_{l-k-(k-h_0)}$ is the LSB of d then we have:

$$\begin{aligned} w_0 2^{k-h_0} + w_1 &= w_0 2^{k-h_0} + dt_1 \\ &= w_0 2^{k-h_0} + (dt_0 - w_0)2^{k-h_0} + (d_{l-k-1}....d_0)_2 \\ &= dt_0 2^{k-h_0} + (d_{l-k-1}....d_0)_2 \\ &= (d_{l-1}....d_0)_2 \\ &= d \end{aligned}$$

7.3 Why the two algorithms above are secure against HMM

As we can notice that there is no where in the algorithm where the value of each bit of the binary representation of the input affects the observable output. Even through the algorithm may look simple or complicated. Therefore:

$$\Pr(\text{output} = D \mid k_i = 0) = \Pr(\text{output} = D \mid k_i = 1) = 1/2$$

and
$$\Pr(\text{output} = A \mid k_i = 0) = \Pr(\text{output} = A \mid k_i = 1) = 1/2$$

So that we have achieved maximum entropy and that means the algorithms are not vulnerable to DPA attack.

8 Flexible Countermeasure using Fractional Window Method

8.1 Original Algorithm

At first a binary representation of an odd integer number is decomposed in a way described by the below algorithm:

```
ALGORITHM VII.11: SPA-resistant width w NAF with Odd Scalar
INPUT: An odd n-bit d = $d_{n-1}...d_0$
OUTPUT: $d_w[n], d_w[n-1], ....d_w[0]$
1. r = 0, i = 0, $r_0$ = w
2. While d > 1 do
   2.1 u[i] = (d mod $2^{w+1}$) - $2^w$
   2.2 d = (d - u[i])/$2^{r_i}$
   2.3 $d_w[r+r_i-1]$ = 0, $d_w[r+r_i-2]$ = 0,..., $d_w[r+1]$ = 0,$d_w[r]$ = u[i]
   2.4 r = r + $r_i$, i = i + 1, $r_i$ = w
3. $d_w[n]$ = 0, $d_w[n-1]$ = 0,.... $d_w[r+1]$ = 0,$d_w[r]$ = 1
4. Return $d_w[n]$,....,$d_w[0]$
```

Proof: d is always odd at every iteration.

$$\begin{aligned}
d - u[i] &= d - [(dmod2^{w+1}) - 2^w] \\
&= d - (dmod2^{w+1}) + 2^w \\
&= A0^{w+1} + 2^w \qquad where\ A = d_{n-1}...d_{w+1} \\
&= A10^w \\
Therefore \\
d &= (d - u[i])/2^w \\
&= A1 \\
&= 1(mod\ 2)
\end{aligned}$$

And then the output sequence $d_w[n]$,....,$d_w[0]$ of the above algorithm will be the input for the Scalar Multiplication as follows.

```
ALGORITHM VII.12: Scalar Multiplication with Width w NAF
INPUT: d = $d_w[n]$,....,$d_w[0]$, P, (—$d_w[i]$—)P
OUTPUT: dP
1. Q = $d_w[c]$P for the largest c with $d_w[c] \neq 0$
2.For i = c - 1 to 0
   2.1 Q = ECDBL(Q)
   2.2 if $d_w[i] \neq 0$ then Q = ECADD(Q,$d_w[i]$P)
2.Return Q
```

Note that this algorithm is secure against both SPA (simple power analysis) and DPA (differential power analysis) because the output sequence always has a fixed pattern:

$$\mid 0^{w-1}X \mid\mid 0^{w-1}X \mid ... \mid 0^{w-1}X \mid \text{ with odd integers } \mid x \mid < 2^w$$

$$\mid D^wA \mid\mid D^wA \mid ... \mid D^wA \mid$$

8.2 Flexible Countermeasure with Fractional Window

The proposed countermeasure using Fractional Window method [8] was proposed by Katsuyuki Okeya and Tsuyoshi Takagi in [9]. For simplicity, we assume the window size is fixed and equal to 4 (w = 4) therefore:

$$\begin{aligned} U_w &= \{\pm1, \pm3, ..., \pm7, \pm9, ..., \pm15\} \\ U_{w-1} &= \{\pm1, \pm3, ..., \pm7\} \\ F &= \{\pm1, \pm3, ..., \pm7, \pm9\} \qquad (Fraction\ window) \end{aligned}$$

The fractional window using class F is constructed by inserting the following step between Step 2.1 and Step 2.2 in Algorithm VII.11 (where B is an integer and 0 < B < 2^{w-1}. In the case of F, we chose B = 1).

If $\mid u[i] \mid > 2^{w-1} + B$ then $u[i] = (u[i] \bmod 2^w) - 2^{w-1}$, r_i = w-1

However the sequence $d_w[n]$,....,$d_w[0]$ generated by this fractional window method has no fixed pattern. For example: we know that $\mid u[i] \mid > 2^{w-1} + B$ if and only if w-2 (r_i - 1) consecutive zeros appear. Otherwise $\mid u[i] \mid \leq 2^{w-1} + B$. As a consequence it is not secure against SPA.

Fortunately, we can get around this problem by two novel ideas suggested in [9] to make the algorithm secure against both DPA and SPA.

• Firstly, we reduce u[i] to the representation of mod 2^{w-1} with the same probability for both $\mid u[i] \mid > 2^{w-1}$ and $\mid u[i] \mid < 2^{w-1}$ by using the following trick:

If $\mid u[i] \mid < 2^{w-1}$ then $u[i] = (u[i] \bmod 2^w) - 2^{w-1}$, r_i = w-1 with Probability 1-P_w.

Where

$$P_w = \frac{\#F - \#U_{w-1}}{\#U_w - \#U_{w-1}} \tag{1}$$

So that the attacker cannot distinguish the two distributions.

• Secondly, elements in the subset F $\backslash U_{w-1}$ are chosen randomly. Assume B = F $\backslash U_{w-1}$. From the above example, we have #F = 10, #U_{w-1} = 8 therefore two elements in set B are chosen randomly from $\{\pm 9, ..., \pm 15\}$ and attackers cannot guess the value of B because of this random choice.

```
ALGORITHM VII.2: SPA resistant fractional window width w NAF with Odd Scalar.
INPUT: an odd n-but d, and a width w (rational number).
OUTPUT: $d_w[n+w_0-1]$,....,$d_w[0]$, and B = $\{\pm b_1, ..., \pm b_{w_1 2^{w_0-2}}\}$
1. $w_0 = \lceil w \rceil, w_1 = w - (w_0 - 1)$
2. Randomly choose distinct integers
   $b_1, ..., b_{w_1 2^{w_0-2}} \in_R (U^+_{w_0} \backslash U^+_{w_0-1}) = \{2^{w_0-1}+1, 2^{w_0-1}+3, ..., 2^{w_0}-1\}$,
   and but B = $\{\pm b_1, ..., \pm b_{w_1 2^{w_0-2}}\}, P_w = w_1$,
   where $U^+_v = \{1, 3, 5, ..., 2^v - 1\}$
3. r = 0, i = 0
4. While d > 1 do
   4.1 x[i] = (d mod $2^{w_0+1}$) - $2^{w_0}$, y[i] = (d mod $2^{w_0}$) - $2^{w_0-1}$
   4.2 If $| x[i] | < 2^{w_0-1}$ then $r_i = w_0, u[i] = x[i]$ with $P_w$,
                                  $r_i = w_0 - 1, u[i] = y[i]$ with $1 - P_w$
         Else if $x[i] \in B$ then $r_i = w_0, u[i] = x[i]$
         Else $r_i = w_0 - 1, u[i] = y[i]$
   4.3 d = (d - u[i])/$2^{r_i}$
   4.4 $d_w[r + r_i - 1]$ = 0, $d_w[r + r_i - 2]$ = 0,..., $d_w[r+1]$ = 0,$d_w[r]$ = u[i]
   4.5 r = r + $r_i$, i = i + 1
5. $d_w[n + w_0 - 1]$ = 0,....,$d_w[r]$ = 1
6. Return $d_w[n + w_0 - 1]$,....,$d_w[0]$, and B = $\{\pm b_1, ..., \pm b_{w_1 2^{w_0-2}}\}$
```

8.3 Hidden Markov Model

Assume that the window size w $\in [4, 5]$, so that there are always at least 4 consecutive Doubling (D) appears. When it comes to the fifth bit we have:

$$\begin{aligned} Pr(w=4 \mid 5^{th}bit=1) &= Pr(w=4 \mid 5^{th}bit=0) &= 1-p \\ Pr(w=5 \mid 5^{th}bit=1) &= Pr(w=5 \mid 5^{th}bit=0) &= p \end{aligned}$$

As you can see, it is possible to draw HMM for this algorithm but it does not mean it is vulnerable to SPA or DPA because at every transition we have:

$$\begin{aligned} Pr(DOUBLING \mid bit=1) &= Pr(DOUBLING \mid bit=0) \\ Pr(ADDITION \mid bit=1) &= Pr(ADDITION \mid bit=0) \end{aligned}$$

As we can not distinguish between zero and non-zero bit so that the algorithm is secure against HMM attacks.

9 Conclusion and Future Work

In this project, I have looked at six randomised exponentiation algorithms, three of them are vulnerable to side channel attacks via IDHMM and another three are not. I also have done some comparisons between two types of IDHMMs proposed in [2] and [3] used to model side channel attacks.

Further work still needs to be carried out. For example: I have not implemented a tool that takes a Smart HMM of an algorithm and a set of output traces as inputs and returns the correspondent secret key. Therefore, it is not clear at the moment, whether or not Smart HMM is actually faster than Karlop-Wagner HMM, even through I can prove that for the same randomised algorithm, Smart HMM tends to be more compact and easier to draw than Karlop-Wagner HMM.

In addition, apart from the six algorithms discussed in this report, I also looked at other randomised algorithms such as Random register renaming to foil DPA proposed by David May in [10], Add and Double always method in [11] and Mongomery Ladder in [11]. However, it is quite trivial that these algorithms are secure against HMM as they always produce fixed pattern output traces.

The conclusion I can draw from here is that Input-Driven Hidden Markov Model can only attack algorithms that work or make decision and choice based on the value of particular bits of the binary representation of a secret key.

References

[1] Liardet,P.Y.., Smart, N..: Preventing SPA/DPA in ECC Systems Using the Jacobi Form. In: Third International Workshop on Cryptographic Hardware and Embedded Systems (CHES).(2001)

[2] Chris Karlop and Davis Wagner: Hidden Markov Model Cryptanalysis

[3] Nigel P Smart: Further Hidden Markov Model Cryptanalysis

[4] Jae Cheol Ha and Snag Jae Moon: Randomised Signed-Scalar Multiplication of ECC to Resist Power Attacks. In: Fourth International Workshop on Cryptographic Hardware and Embedded Systems (CHES).(2002)

[5] Colin D.Walter: Mist: An Efficient, Randomized Exponentiation Algorithm for Resisting Power Analysis. In: RSA 2002-Cryptographers' Track. (2002)

[6] Kouichi Itoh, Jun Yajima, Masahiko Takenaka and Naoya Torii: DPA Countermeasure by Improving the Window Method. In: Fourth International Workshop on Cryptographic Hardware and Embedded Systems (CHES).(2002)

[7] Colin D.Walter: Breaking the Liardet-Smart Randomized Exponentiation Algorithm. In: Fifth Smart Card Research and Advanced Application Conference (CARDIS). (2002)

[8] Bodo Moller: Improved Techniques for Fast Exponentiation. In: the fifth International Conference on Information Security and Cryptography (ICISC 2002) LNCS 2578, (2003), 298-312

[9] A More Flexible Countermeasure against Side Channel Attacks Using Window Method

[10] D. May, H.L. Muller and N.P Smart: Random Register Renaming to Foil DPA. In: Third International Workshop on Cryptographic Hardware and Embedded Systems (CHES).(2001)

[11] Kouichi Itoh, Tetsuya Izu and Masahiko Takenaka: A Practical Countermeasure against Address-Bit Differential Power Analysis

[12] I.F.Blake, G.Seroussi and N.P.Smart. Elliptic Curve in cryptography. Cambridge University Press, 1999.

[13] Elisabeth Oswald and Manfred Aigner: Randomized Addition-Subtraction Chains as a Countermeasure against Power Attacks. In: Third International Workshop on Cryptographic Hardware and Embedded Systems (CHES).(2001)